VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

DIPLOMOVÁ PRÁCE

2010 Bc. Jiří Gabriel

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Systém pro správu definic křižovatek
Crossroad Definitions Management System

2010 Bc. Jiří Gabriel

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 4. května 2010 . . . . . .

podpis

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu diplomové práce, panu Ing. Janu Martinovičovi Ph.D., za podmětné připomínky při tvorbě této práce.

Abstrakt

Cílem této diplomové práce je navrhnout a implementovat aplikaci zajišťující snadnou správu projektů světelné signalizace pro řízení světelných křižovatek. Tato aplikace umožní uživateli přehledně organizovat projekty po jednotlivých městech a křižovatkách. V rámci křižovatky pak vést jednotlivé verze projektu s grafickým odlišením. Dále pak zajišťuje editaci vlastností řadiče světelné signalizace a přípravu dat pro odeslání do řadiče. Dokument nejprve vysvětluje základní pojmy z oboru dopravní inženýrství. V další kapitole se nachází text věnovaný specifikaci požadavků na vytvářenou aplikaci. Dále je popsán návrh implementace, samotná implementace částí systému a technologie použité při vývoji. Nakonec je popsána distribuce a nasazení u uživatele.

Klíčová slova

.NET Framework, XML, C#, SSZ, CROSS PTC

Abstract

The aim of this work is to design and realize application for the easy management of the traffic lights projects to control traffic light intersection This application allows users to neatly organize project by individual cities and intersections. The intersection will include different versions of a project with graphic modulation. Application provides editing features controller of traffic lights and prepare to send data to controllers. Document firstly explains the basic concepts of the field of traffic engineering. The next chapter text is devoted to specification requirements for the building application. The next chapter describe design application, own implementation of the system parts and technologies used in developing this system. Finally, describe distribution and deployment of the user.

Keywords

.NET Framework, XML, C#, SSZ, CROSS PTC

Seznam použitých symbolů a zkratek

.NET	- Aplikační platforma společnosti Microsoft
CSV	- Comma-separated values (hodnoty oddělené čárkami)
DLL	- Dynamic-link library (dynamicky spojená knihovna)
INI	- Configuration files (konfigurační soubor)
TXT	- Text file (textový soubor)
XML	- eXtensible Markup Language (rozšiřitelný značkovací jazyk)
XSD	- XML Schema Definition (Definice XML schématu)

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Řízení dopravy pomocí světelného signalizačního řízení se neustále vyvíjí. Je tedy nutné aplikace a řadiče pro podporu řízení světelné signalizace vylepšovat, popřípadě vyvinout nové a lepší.

Cílem této diplomové práce je navrhnout a implementovat aplikaci pro snadnou správu definic křižovatek ve městech. V rámci křižovatky pak vést jednotlivé verze jejich definic s grafickým odlišením aktivní, archivní a připravované verze. Na pozadí tohoto systému vést totožný stromový systém složek. Vyvíjená aplikace se bude jmenovat CROSS PTC a dále bude sloužit pro parametrické programování a import dopravních definic vytvořených v jiných programech. Dále zajišťuje editaci vlastností řadiče světelného signalizačního zařízení a přípravu dat pro odeslání do řadiče. Aplikace bude určena pro operační systém Microsoft Windows. Veškerá nadefinovaná data budou ukládána ve formátu XML.

1.1 Struktura práce

Ve 2. kapitole této práce budou vysvětleny základní pojmy z oboru dopravního inženýrství (z oblasti světelná signalizační zařízení) týkající se této aplikace. Specifikaci požadavků vytvářené aplikace se bude věnovat 3. kapitola. Zde budou popsány jednotlivé funkce systému, a kdo s ním bude pracovat. Ve 4. kapitole (Návrh implementace) bude popsána architektura aplikace. V 5. kapitole (Implementace) se věnuji podrobnému popisu způsobu implementace jednotlivých částí aplikace a použitých technologií při vývoji. V 7. kapitole (Ukázka aplikace) jsou zobrazeny obrázky vytvořené aplikace. V 7. kapitole (Testování) bude popsán postup testování při vývoji aplikace. Způsob nasazení a distribuce aplikace k zákazníkovi (uživateli) je popsán v 8. kapitole.

2 Dopravní inženýrství

Dopravní inženýrství je velmi rozsáhlý a členitý obor. Jedním z oborů dopravního inženýrství je projektování světelné signalizace, na které pak navazuje činnost v oblasti realizace světelného signalizačního zařízení (SSZ). Právě pro tyto činnosti je určena aplikace, která je předmětem této diplomové práce.

Dopravní inženýři jsou osoby zabývající se oborem dopravního inženýrství, v našem případě konkrétně návrhem SSZ. Výsledkem jejich práce je projekt světelně řízené křižovatky nebo přechodu pro chodce a cyklisty.

Takto vytvořený projekt se skládá ze:

- Stavební části – jedná se o terénní úpravy a samotnou stavbu křižovatky.
- Části elektrických rozvodů a výstroje (sloupy, návěstidla, rozvodné krabice, řadič SSZ) – montáž elektrických částí potřebné pro realizaci SSZ.
- Dopravního řešení (řadicí pruhy, použité signály, signální plány) – vytvoření definic pro samotný chod SSZ, k těmto účelům bude sloužit vyvíjená aplikace.

2.1 Historie

První elektrická světelná signalizace v Evropě byla postavena v roce 1924 na Postdamer Platz v Berlíně. Byla to 8,5 metru vysoká pětiboká věž s kabinou v horní části. Barevné signály byly umístěny nad kabinou a jednotlivé barvy byly vodorovně vedle sebe. Jednotlivé fáze řízení dopravy přepínal ručně policista sedící v kabině [14].

Na území tehdejšího Československa bylo první SSZ postaveno v roce 1927 na ulici Hybernská, Dlážděná a Havlíčkova u dnešního Masarykova nádraží v Praze [14].

2.2 Světelná signalizační zařízení (SSZ)

Světelné signalizační zařízení je název pro systém skládající se ze soustavy zařízení napomáhající k zabezpečení řízení a bezpečnosti provozu na pozemních komunikacích pomocí světelných signálů. V běžném životě se s ním setkáme na křižovatkách, dálnicích, tunelech, ale i při opravách silnic. V obecné češtině se pro viditelnou část tohoto zařízení používá název semafor.

Na rozdíl od jiných metod řízení dopravy je SSZ schopno rychle reagovat na potřeby dopravy a přerozdělovat priority mezi účastníky silničního provozu. Tato schopnost je velmi důležitá pro zajištění maximálně plynulého a bezpečného provozu na komunikacích, hlavně ve městech.

V principu se jedná o rozmístění návěstidel (semaforů), tak aby byly řízeny jednotlivé řadicí pruhy křižovatky a přechody pro chodce pomocí střídání signálů povolujících a zakazujících vjezd na pozemní komunikaci. Návěstidla se propojují napájecími kabely, které jsou všechny vedeny do řadiče SSZ, který je vybaven logikou, zajišťující bezpečné střídání signálů podle dopravního řešení [13].

2.3 Signály

Velmi důležitou roli při řízení dopravy pomocí SSZ hrají signály. Jsou to dohodnuté symboly pro zobrazení informací účastníkům silničního provozu. Na jejich znalosti, hlavně ze strany účastníků dopravy, je postaveno celé řízení pomocí SSZ. Liší se tvarem a významem podle zvyklostí státu. Soubor signálů určených jedné skupině účastníků provozu se nazývá signální skupina. Posloupnost a délka trvání signálů je dána signálním plánem.

Význam signálů a související povinnosti účastníků provozu stanoví § 70 až § 74 zákona č. 361/2000 Sb., o silničním provozu, v platném znění. Podle § 24 a přílohy vyhlášky 30/2001 Sb. se užívají tyto signály [11][12]:

- S1 Tříbarevná soustava s plnými signály.
- S2 Tříbarevná soustava se směrovými signály.
- S3 Tříbarevná soustava s kombinovanými směrovými signály.
- S4 Signál žlutého světla ve tvaru chodce (umístěná vedle signálu se zeleným směrovým signálem pro odbočení, případně i před přechodem, jehož se týká) upozorňuje řidiče, že křižuje směr chůze přecházejících chodců. Nepřerušovaně i přerušovaně svítící signál má stejný význam.
- S5 Doplňková zelená šipka (umístěná vedle signálu s červeným světlem Stůj!) – svítí-li současně s červeným nebo žlutým světlem, umožňuje pokračovat v jízdě příslušným směrem, řidič však musí dát přednost vozidlům ve volných směrech a nesmí omezit ani ohrozit přecházející chodce.
- S6 Signál pro opuštění křižovatky (tzv. vyklizovací šipka, umístěná v protilehlém roku křižovatky)
- S7 Přerušované žluté světlo.
    - Signál s červeným světlem Stůj! znamená povinnost zastavit vozidlo před příčnou čarou, a kde taková čára není, před signalizačním zařízením. V případě použití dvoubarevné soustavy (bez žlutého světla) má červené světlo stejný význam jako samostatně svítící žluté světlo.
    - Signál se současně svítícím červeným světlem a žlutým světlem Pozor! znamená povinnost připravit se k jízdě.
    - Signál se zeleným světlem Volno znamená možnost pokračovat v jízdě. Jde-li o signál se směrovým signálem nebo svítí-li pro odbočení vlevo Signál pro opuštění křižovatky, není třeba při příslušném odbočování dávat přednost protijedoucím vozidlům a souběžným tramvajím. Jde-li o signál Volno! se směrovým signálem a nesvítí-li současně signál žlutého světla ve tvaru chodce, není třeba při odbočování dávat přednost chodcům v kolizním směru.

- Signál s nepřerušovaně svítícím žlutým světlem Pozor! má obdobný význam jako signál Stůj!. Je-li však při rozsvícení signálu vozidlo tak blízko, že by řidič nestačil zastavit, smí pokračovat v jízdě.
- Svítí-li žluté světlo Pozor! přerušovaně, nejde momentálně o křižovatku s provozem řízeným světelnými signály. Je-li signál užit společně s dopravní značkou nebo dopravním zařízením, zdůrazňuje jejich význam. Je-li použit samostatně, upozorňuje na nutnost dbát zvýšené opatrnosti.
- K zabezpečení vjezdu na pozemní komunikaci se někdy používá návěstidlo pouze s červeným a žlutým světlem (bez zeleného).

- S8a Zakázaný vjezd vozidel do jízdního pruhu.
- S8b Volný vjezd vozidel do jízdního pruhu.
- S8c Světelná šipka vlevo, S8d Světelná šipka vpravo – přikazují opuštění jízdního pruhu nebo objetí překážky příslušným směrem.
- S8e Světelný kříž – označuje překážku provozu vedle vozovky.
- S12 Rychlostní signály s proměnným signálním znakem nebo s více signálními znaky, vyznačují doporučenou rychlost v km/h.
- S13 Signál dvou vedle sebe umístěných přerušovaných červených světel – má obdobný význam jako červené světlo dvoubarevné soustavy.
- S14 Signály přejezdového zabezpečovacího zařízení.

2.4 Návěstidlo

Teď, když už víme, jaké jsou signály, je zapotřebí tyto signály nějak zobrazovat. Pro tento účel nám slouží návěstidlo, které má za úkol zobrazovat signály účastníkům provozu. Je složeno ze světelného zdroje, optické soustavy, nosné konstrukce a elektronické výzbroje. Může se lišit svým provedením, ale vždy musí splňovat požadavek správného zobrazení signálu. Volba světelného zdroje má významný vliv na kvalitu a čitelnost signálu. Jsou použity žárovky nebo LED diody o napájecím napětí 230, 40 nebo 10V.

2.5 Řadič SSZ

Aby SSZ mohlo jako celek fungovat, je nutno dodat „mozek“ celého zařízení a to je řadič. Řadičem SSZ nazýváme zařízení, které rozhoduje o signálním obrazu křižovatky v reálném čase. Řadič je konstruován jako bezpečné zařízení, je tedy nutné, aby v případě vlastní poruchy nebo poruchy venkovního vybavení zareagoval bezpečně a předešel možné dopravní nehodě [13].

Funkce řadiče

- Realizace logiky řízení v SSZ – podměty k spínání příslušných obvodů signálů

- Kontrola bezpečnostních atributů funkce – porušení časových a funkčních závislostí
- Zajištění zobrazení korektních signálů – napájení světelných zdrojů návěstidel
- Řízení detektorů – funkce pro dynamické a adaptivní řízení
- Řízení poruch – zachování funkce řadiče při vzniku poruchy, která neovlivní bezpečnost provozu
- Diagnostické nástroje – prostředky pro určení zdroje poruchy ovlivňující funkci zařízení
- Záznam dat o dopravě – vyhodnocení efektivity použité logiky řízení

2.6 Signální plán

Zdrojem informací pro řadič SSZ jsou signální plány. Signální plán definuje sekundu po sekundě, jakým způsobem se bude vyvíjet signální obraz křižovatky po jedno opakování. Je složen z fází, ve kterých jsou sdruženy signály popsané v kapitole 2.3 a fázových přechodů. Signální plán popisuje posloupnost spínání signálů a délku jejich sepnutí. Fází se rozumí část signálního plánu, kde nedochází ke změně signálu. Fázové přechody jsou oblasti, kde dochází ke změně signálů mezi fázemi. Pokud se jedná o dynamické řízení, signální plán pak definuje meze parametrů pro logiku řízení. Tím je zabezpečeno, aby se signální plán přiblížil co nejvíce reálným požadavkům účastníků provozu [13].

2.7 Dopravní řešení

Dopravní řešení je dopravním inženýrem vytvořené zpracování dopravního řešení. Dalo by se říct, že se jedná a popis fungování uzlu (komunikace) vycházející z norem daného státu, které jsou harmonizovány. Každý stát má však svou národní přílohu, která zohledňuje místní specifické podmínky. Některé podklady jsou dány nejen normou, ale i zákonem nebo vyhláškou (např. úprava provozu na veřejných komunikacích). Umožňuje pomocí standardizovaných pojmů, veličin a specifikací popsat vztahy v posuzovaném uzlu tak, aby byly objektivně popsatelné a pochopitelné.

Optimální dopravní řešení vykazuje nejnižší míru omezení vyjádřenou stupněm kvality plynulosti dopravy a nejvyšší hodnotu rezervy kapacity pro všechny vjezdy uzlu. Stupeň kvality plynulosti dopravy je určen zdržením účastníka dopravy při průjezdu uzlem. Cílem je, aby toto zdržení bylo co nejmenší. Rezerva kapacity definuje procentní rozdíl mezi poptávkou a kapacitou vjezdu uzlu [13].

Hlavní důraz norem je kladen na bezpečnost. Normy stanoví podmínky, které vedou k zajištění bezpečnosti řidičů a chodců na křižovatkách se SSZ. Omezují i seznam použitých materiálů a zařízení a podmínky, za kterých se smí používat (certifikované řadiče SSZ, světelné zdroje ap.).

2.8 Detektory

Dalším prvkem SSZ jsou detektory, pomocí nichž může SSZ reagovat na aktuální dopravní potřeby při řízení provozu. Jedná se o technické zařízení, které je schopno v reálném čase poskytovat data o dopravě z definovaného místa. Proces získávání dat se nazývá detekce. Její výsledky jsou použity k případné modifikaci signálního plánu v reálném čase a tím reagovat na aktuální potřeby účastníků provozu. Detekce taky získává statistické data o dopravě pro vyhodnocení vhodnosti nasazení algoritmů dopravního řešení [13].

Typy detektorů

- Kontaktové detektory – jedná se o tlačítka pro chodce a jiné, založené na principu mechanického spojení elektrického kontaktu.
- Indukční detektory – jedná se indukční smyčky zařezané v povrchu vozovky reagující na kovovou konstrukci vozidla. Výhodou je jednoznačně definovaná detekční zóna, spolehlivost a přesnost.

Pokud však není technicky možné zapracovat indukční detektory do vozovky, používají se detektory na jiných technických principech.

- Video detektory – video procesor zpracovává informace na detekce vozidel čekajících na křižovatce nebo blížících ke křižovatce. Výstupem jsou standardní dopravní data. Spolehlivost může být ovlivněna povětrnostními podmínkami.
- Infračervené detektory – rozpoznávání přítomnosti vozidla pomocí jeho vyzařování v infračerveném pásmu. Detekční zóna je vytvářena virtuálně.

Jsou vytvářeny i speciální detektory založené na principu přenášení dat z vozidel městské hromadné dopravy (MHD) nebo z vozidel s předností jízdy.

- Selektivní detektory IZS – zajišťují detekci přítomnosti vozidel v proudu jiných vozidel, včetně polohy vozidla v profilu komunikace. Jedná se většinou o detektory s úzce směrovou charakteristikou a schopností obousměrné komunikace mezi vozidlem IZS a systémem SSZ.
- Selektivní detektory MHD – zajišťují detekci vozidel MHD. Kromě samotné detekce vozidla mohou identifikovat konkrétní parametry jízdy.

Všechny informace z detektorů se shromažďují v řadiči, kde jsou použity pro reakci na aktuální dopravní situaci a požadavky.

2.9 Postup vzniku SSZ

Postup vzniku celé křižovatky lze zjednodušeně sepsat do následujících bodů.

- Křižovatka – výchozí podklady - AUTOCAD, plány.
- Dopravní sčítání – počet vozidel během dne v průběhu týdne + rozdělení podle kategorií, (osobní, nákladní, dodávky), sčítání se provádí ručně nebo automatizovaně.
- Zohlední se záměr investora, policie, místní podmínky, psychologie řidičů, další faktory, (otevření nákupního střediska ap.) – jednání o návrhu. Mohou být různé priority, zlepšení, Dopravy, snížení nebezpečnosti ap.
- Definice pruhů, stop čar a signálů.
- Výpočet mezičasů.
- Výpočet kapacit.
- Návrh fázování, minimálních (sedlových) a maximálních (špičkových) plánů.
- Výpočet koordinace.
- Definice požadavku na řadič SSZ – počet světel, vstupů, výstupů a specifických parametrů (dopravně proměnné značky, informační tabule, propojení s přejezdy pro vlaky, tunelem, apod.).
- Realizace – stavební úpravy, práce na elektrických rozvodech, příprava detektorů, montáž návěstidel, řadiče.
- Pro řadič SSZ vzniká konečný produkt, dopravní řešení, podle kterého bude SSZ pracovat. Vznikají tak především signální plány.

2.10 Projektování signálních plánů

Signální plány popisované v kapitole 2.6 potřebné pro řadič SSZ, lze definovat dvěma způsoby popsanými níže.

2.10.1 Standard RILSA

Definice křižovatky je vytvořena pomocí fází, fázových přechodů a vývojového diagramu. Nevýhodou tohoto standardu je, že se definují dopravní algoritmy vždy znovu a mají určitá omezení ve svých možnostech (proměnná skladba fází, uplatnění mezičasů přes více fází apod.). Tento způsob definice je časově náročnější. Definice je ale obecně přenositelná na řadiče více výrobců. Nevýhody se kompenzují tím, že se jednou hotová řešení kopírují. Omezení v tvorbě plánů se obchází speciálními funkcemi, které jsou ale nestandardní.

2.10.2 Parametrické programování

Dopravní algoritmy jsou předdefinované, definují se jen parametry, které je upravují. Definice je pak mnohem rychlejší, algoritmy jsou více otestované, takže i doba testování řešení je kratší. Parametry obvykle definují řazení jednotlivých fází signálního plánu, délku jejich trvání a podmínky, které řazení a délku ovlivní. Nevýhodou je, že SW pro parametrické definice je vždy na konkrétní typ řadiče a pokud je jen parametrický, tak není možné doplnit jiné dopravní algoritmy.

2.11 Řadič SSZ CROSS RS4

CROSS RS4 je řadič světelného signalizačního zařízení vyvinutý firmou CROSS Zlín s.r.o. K čemu slouží řadič SSZ jsem si popsali v kapitole 2.5. Tento řadič umožňuje programování pomocí standardu RILSA (kapitola 2.10.1) pomocí návrhové aplikace LISA firmy Schlothauer & Wauer Berlin (kapitola 0). Dále lze provést parametrické programování (kapitola 2.10.2) pomocí sady parametrů, které se aplikují na algoritmy, vyvíjené po dobu 10 let. Parametrické programování je doplněno volně programovatelnou částí v jazyce C, takže kromě parametrů umožní i dopsat modifikace dopravních algoritmů.

2.12 Podporovaný software

Vyvíjení aplikace CROSS PTC spolupracuje s následujícími dvěma programy. Vytvořené definice v aplikaci LISA+ bude možno importovat do naší aplikace a provádět s ní další operace. CROSS PTC bude obsahovat rozhraní pro komunikaci se softwarem CROSS LTC, který se stará o celkový monitoring.

LISA+

Jedná se o dopravně inženýrský software sloužící pro navrhování a řízení světelné signalizace vyvinutý firmou Schlothauer & Wauer. Aplikace dovoluje návrh signálních plánů pomocí grafického rozhranní a testování na reálných hodnotách pomocí simulátoru funkcí. Umožňuje přípravu dat pro řadiče světelné signalizace. Aplikace provede uživatele celým procesem návrhu světelné signalizace od sběru dat až po přípravu programu pro řadič.

CROSS LTC

Jedná se o aplikaci vyvinutou firmou CROSS Zlín s.r.o. určenou pro firmy zabývající se poskytováním služeb pro světelná signalizační zařízení jako je sledování jejich stavu a vykonávání jejich údržby. Software umožňuje jak vzdálenou tak lokální správu řadiče SSZ jako je například časový harmonogram, signální plány a intenzitu dopravy.

3 Specifikace požadavků

Jedná se o úvodní etapu při návrhu systému. Jsou zde definovány hlavní funkce systému, a kdo bude se systémem pracovat. Většinou slouží k odsouhlasení zadání mezi vývojovým týmem a zadavatelem.

3.1 Specifikace zadání

Aplikace CROSS PTC umožní uživateli přehledně organizovat křižovatky ve městech. V rámci křižovatky pak vést jednotlivé verze jejich definic s grafickým odlišením aktivní, archivní a připravované verze. Na pozadí tohoto systému bude stromový systém složek reprezentující jednotlivá města, křižovatky a verze. Součástí aplikace bude možnost kopírování verzí a jejich export a import ve tvaru XML. Bude umožněno ukládání kompletní definice křižovatek ve formátu XML.

3.2 Kdo bude se systémem pracovat

Se systémem budou pracovat uživatelé rozdělení do několika skupin dle oprávnění pro editaci definic. Do aplikace neověřený uživatel nebude mít přístup.

3.3 Funkční specifikace systému

Systém bude obsahovat pouze interní část, která bude přístupná po zadání přihlašovacích údajů.

V systému budou implementovány následující funkce:

1. Přihlášení do aplikace
2. Správa křižovatek ve městech
3. Editace
    a. definice řadiče
    b. GSM centrály
    c. sekcí
    d. signálních skupin
    e. napájení
    f. dopravních detektorů
    g. tabulky mezičasů
    h. tabulky kolizí
    i. parametrů signálních plánů
    j. fází
    k. rozvrhů lokálního řízení
    l. výstupů
    m. preference MHD
    n. dalších pomocných údajů
4. Možnost vícejazyčné verze aplikace
5. Tisk dat ve formě textového dokumentu, grafický tisk navržených signálních plánů

6. Zabezpečení definice v XML proti přepsání jinými prostředky nebo ruční editací
7. Kompilace volné programové části
8. Import z aplikace LISA+
9. Generování definice pro upload
10. Editace volně programovatelné části
11. Prohlížení zdrojových souborů
12. Testační kompilace pro PC
13. Generování bezpečnostního kontroléru
14. Načítání historie z bezpečnostního kontroléru

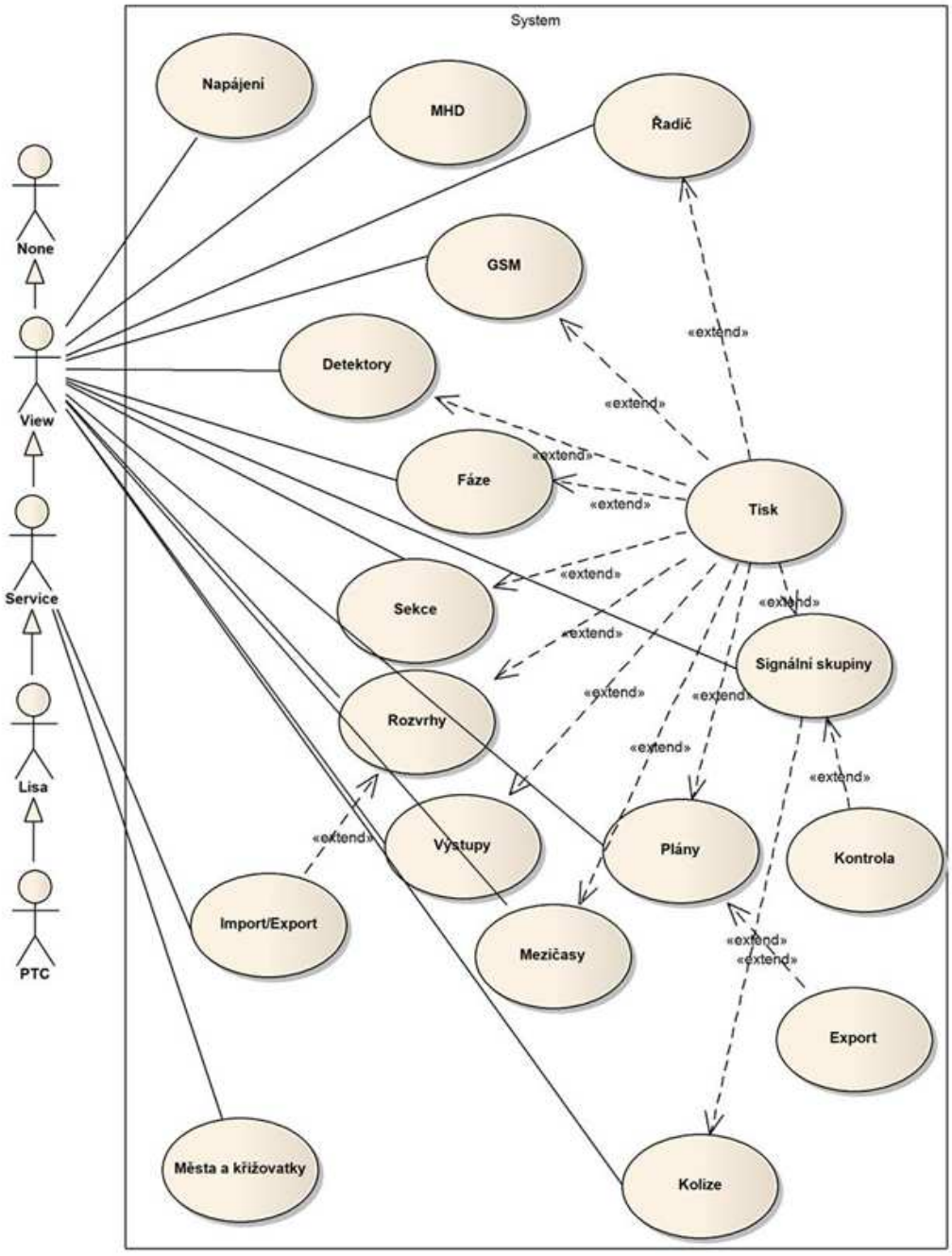
3.4 Požadavky firmy

Firma CROSS Zlín preferuje snadné a rychlé nasazení aplikace u živitele, dále snadnou konfiguraci a přenositelnost definic a konfigurace aplikace mezi počítači. Z těchto důvodů jsou stanoveny následující požadavky na vytvářený software.

1. Operační systém Windows XP a výše.
2. Některé PC, na kterých bude aplikace instalována, nejsou připojeny z bezpečnostních důvodů nikdy do žádné sítě nebo internetu, nemusí na nich být aktualizace. Je možné provést aktualizaci v okamžiku instalace aplikace, ale pak už dále ne. Aplikace i všechno další se na tyto počítače může někdy instalovat jen přes USB disk.
3. Programovací jazyk není stanoven, preferujeme použití MVS C#.
4. Nepoužívat registry OS Windows, vše ukládat do INI souborů v adresáři aplikace.
5. Minimum „toulání“ po pevném disku počítače, vše soustředit do jedné složky.

3.5 UseCase

Use case, neboli diagram případů užití slouží pro grafické znázornění jednotlivých částí specifikace požadavků systému. Více o specifikaci UML se nachází v dokumentu [3].

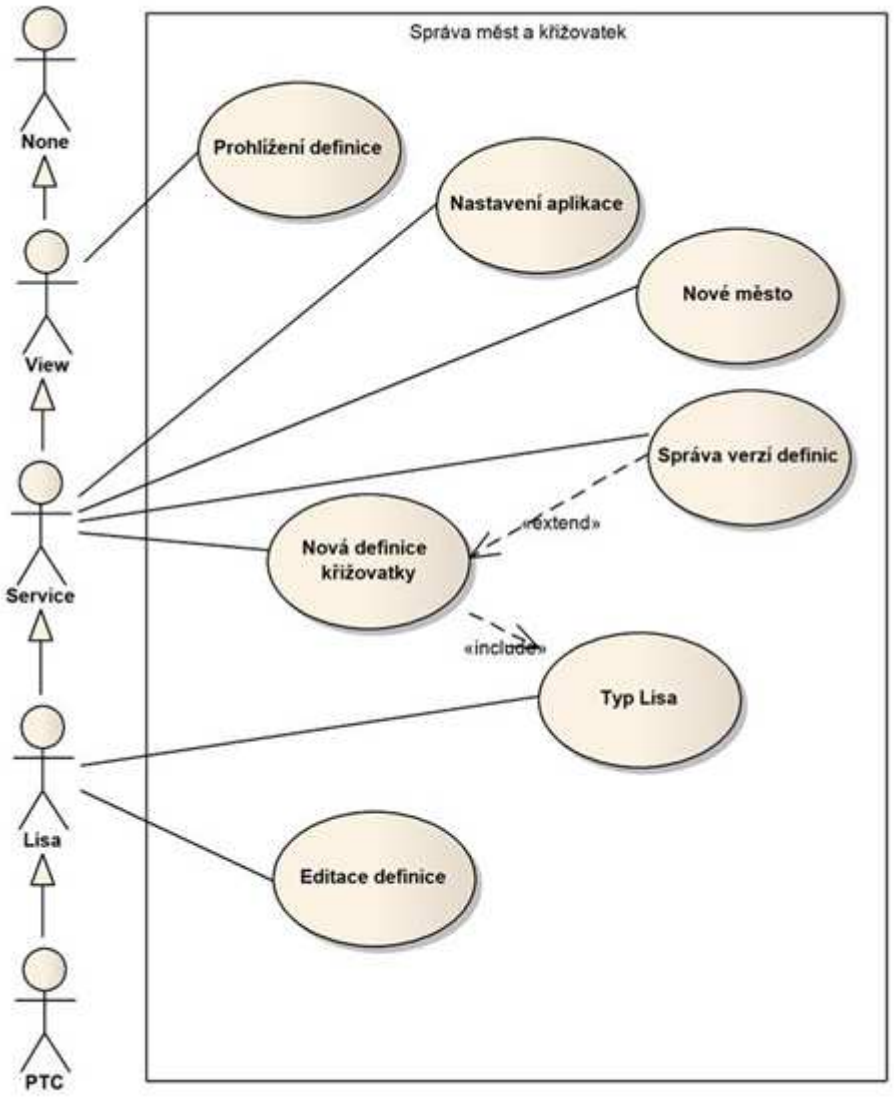


Obrázek 1: Use case – Systém

Na obrázku 1 je znázorněn diagram případů užití popisující funkčnost z pohledu celého systému. Na levé straně obrázku jsou znázorněny role uživatelů. Každá role má povolen přístup jen

k funkcím systému specifickým ke každé roli. Na obrázku jsou zachyceny všechny funkce systémů a jejich rozšíření.

Další diagram případů užití na obrázku Obrázek 2 popisuje chování správy měst a křižovatek. Opět je graficky znázorněno oprávnění jednotlivých uživatelů k funkcím systému. Správa měst a křižovatek slouží pro snadnou grafickou správu definic křižovatek ve městech a jejich verzí.



Obrázek 2: Use case – Správa měst a křižovatek

3.6 Slovní popis funkcí

V této podkapitole budou podrobně rozepsány jednotlivé funkce aplikace. Z důvodu zachování interních informací firmy CROSS Zlín s.r.o. nebudou funkčnosti popisovány podrobně. Veškerá definice se provádí pomocí interní dokumentace.

Všechny položky definice musí být možno editovat a pak přenášet do definice pro UPLOAD do řadiče. Aplikace musí počítat s možností doplňování dalších vlastností do budoucna s dodržením zpětné kompatibility. Přidání nové položky nesmí způsobit stav, kdy starší definice budou nepoužitelné.

3.6.1 Přihlášení

Po spuštění aplikace je uživatel vyzván k zadání uživatelského jména a hesla do systému. Aplikace je funkční pouze po úspěšném ověření uživatele, obsahuje tedy pouze interní část. Přihlášení musí být provedeno bez využití databázového serveru a internetu.

Správa hesel nesmí být na uživateli. Dodavatel aplikace požaduje, aby byl vytvořen centrální seznam uživatelů a jejich hesel (důvodem je evidence všech oprávněných osob a informace o autorizovaných přístupech k řadičům SSZ). Přihlášení nesmí být vázáno na počítač. Na jednom počítači se může přihlašovat i více osob. Heslo musí být uživatelem zadáno, nesmí se využívat aparátu Windows pro ukládání hesel.

3.6.2 Správa křižovatek ve městech

Formulář pro správu křižovatek ve městech bude současně sloužit jako hlavní okno aplikace. Po spuštění se zobrazí formulář pro přihlášení do aplikace. Po úspěšném zadání přihlašovacích údajů se okno pro správu křižovatek odemkne.

Každý uživatel bude mít možnost přidat města, křižovatky a následně definice křižovatek (řadičů) ve městech, které jsou spravovány ve třech následujících verzích:

- Aktuální – určena k distribuci do řadiče a pro monitorování stavu řadiče
- Archivní - rozpracovaná verze, v níž jsou chystány nějaké změny
- Pracovní - byla použita v minulosti, je uložena jen pro evidenci. Není v ní možné editovat, pouze prohlížet

Po vytvoření města a křižovatky nejsou k dispozici žádné verze definic. Vytvořená nová definice je automaticky nastavena jako pracovní, kterou lze odstranit, převést na verzi aktuální nebo vytvořit novou pracovní verzi z vybrané pracovní verze. Aktuální verze může být vždy v aplikaci pouze jedna. Lze ji zkopírovat do archivu nebo z ní vytvořit novou pracovní verzi. Při nastavování nové aktuální verze se musí současně nastavená aktuální verze nejprve archivovat a následně se pracovní verze přesune jako aktuální. Z archivní verze lze vytvořit nová pracovní verze.

Souborově jsou verze organizovány tak, že v adresáři křižovatky je umístěna aktuální verze. Při vzniku pracovní verze je vytvořen adresář WORKING_číslo, při vzniku archivní verze pak ARCHIVE_číslo. Čísla jsou přidělována postupně od 1, takže správně číslované archivy mají vzestupnou řadu, u pracovních verzí může být číslování různé, protože pracovní verze vznikají a zanikají libovolně.

Při vzniku archivní verze je vždy kopírován celý obsah adresáře. U pracovní verze pouze soubory XML, c a h.

3.6.1 Import z LISA+

Akce zajistí zpracování údajů, editovaných v programu LISA+. Při prvním importu dojde ke kompletnímu přenosu definice, při dalších importech jsou již zohledněna data jednou načtená, aby se zachovaly již editované objekty. Pokud je řadič definován pomocí importu z aplikace LISA + je nutné zakázat editaci některých položek.

3.6.2 Definice řadiče pro UPLOAD

Aplikace vytvoří sadu souborů s popisem řadiče na zadaném disku. Pokud je uložení nasměrováno na USB disk, lze ho přímo použít pro přenesení do řadiče RS4. Před vytvořením definice se musí provést kompilace volné programové části. Tím se zajistí, aby k definici byla přiložena i programovatelná knihovna.

3.6.3 Kompilace volné programovatelné části

Provede překlad zdrojového kódu dynamiky řadiče a zobrazí pomocí nastaveného editoru výsledek překladu. Pokud jsou v překladu chybová hlášení, v žádném případě se nesmí odeslat definice do řadiče!

3.6.4 Editace volně programovatelné části

Řadič obsahuje kus kódu, napsaného v programovacím jazyce C, který potřebujeme, aby se dostal do řadiče a tam slinkoval s firmwarem a byl funkční. Překlad musí být pro procesor ARM s použitím překladače GCC. Nesmí se přenášet, ani dostat do řadiče přímo zdrojový kód. Musí být možné provést kompilaci přímo z aplikace, aby bylo vidět výsledek a neposílal se do řadiče vadný kód.

3.6.5 Prohlížení zdrojových souborů

Dává uživateli možnost prohlédnout si zdrojové kódy. Nejprve se provede výběr souboru, v dalším kroku se pak pomocí předvoleného editoru zobrazí vybraný soubor.

3.6.6 Testovací kompilace pro PC

Tato možnost se používá v případě, kdy se připravují soubory k odeslání na testovací definice řadiče na PC.

3.6.7 Bezpečnostní kontrolér

Určeno k naprogramování bezpečnostního kontroléru (SWC), který monitoruje a dohlíží na procesorovou jednotku ARM.

3.6.7.1 Editace tabulky kolizí

Tato tabulka je uložena v SWC ke kontrole bezpečnostních mechanismů, nemusí být totožná s tabulkou mezičasů.

3.6.7.2 Generování bezpečnostního kontroléru

Pomocí této akce se generuje soubor HEX zaveditelný do SWC obsahující definici nutnou pro bezpečnostní funkce. Generování se liší podle typu řadiče (vše ostatní je zcela společné).

3.6.7.3 Načtení historie z bezpečnostního kontroléru

Pomocí této funkce dojde k načtení historie z bezpečnostního kontroléru a následnému uložení do souboru.

3.6.8 Definice řadiče

Zde budou definovány podrobné informace potřebné pro funkčnost řadiče SSZ. Údaje budou editovány v rozsahu a struktuře, která bude předána v neveřejném dokumentu „Datová struktura řadiče RS4“.

3.6.1 GSM definice

Definuje odesílání SMS zpráv pro servisní pracovníky nebo pro účely monitoringu přes centrálu. Všechna čísla se vkládají včetně směrovacího kódu země. Rozsah odesílání SMS je dáno definicí každého čísla v definici GSM řadiče, pokud nemá číslo nic zatrženo, nic se neodesílá.

Uživatel bude mít možnost definovat nová čísla, editovat nebo mazat stávající.

3.6.2 Sekce

Rozdělení na sekce má význam při ovládání více částí křižovatek nebo přechodů pro chodce, které jsou na sobě dopravně závislé, ale nemají mezi sebou mezičasy. Sekce umožňují samostatné vypnutí do kmitavé žluté nebo samostatný chod sekce při poruše světla jiné sekce. Sekce jsou pak použity v definici signální skupiny.

Uživatel bude mít možnost přidat nové sekce, editovat nebo mazat stávající sekce.

3.6.3 Signální skupiny

Signální skupina je soubor signálů určených jedné skupině účastníků provozu, které jsou pak zobrazovány pro řízení dopravy. Tyto skupiny jsou pak použity při definici detektorů, tabulky mezičasů a kolizí, MHD a plánů.

Uživatel bude mít možnost definovat nové signální skupiny, editovat nebo mazat stávající skupiny. Formulář bude rovněž poskytovat funkci kontroly zadaných údajů, zda nedochází ke kolizím mezi skupinami. Dále bude obsahovat zpracování příkonů žárovek jak individuálně, tak hromadně a import těchto hodnot ze souboru.

3.6.4 Napájení

Pro jednotlivé použité spínačové desky (rozlišené adresami) se zde vybírá napájecí napětí s možnostmi 230, 40 a 10 V, což samozřejmě musí být v souladu se skutečným napájecím napětím. Nabízí se jen definované adresy spínačových desek, dokud nejsou definovány výstupy světel u signálních skupin, tak se nic nenabízí. Světla v návěstidlech jsou buď žárovky, nebo LED diody.

Uživatel pro každou adresu vybere úroveň napájecího napětí žárovky. Uživateli budou zobrazeny jen definované adresy.

3.6.5 Detektory

Detektor je vstupní zařízení, na základě kterého se získávají informace, sloužící k modifikacím signálních plánů. Jedná se fyzicky o smyčkové indukční detektory, chodecká tlačítka, jiné detektory (IR, radary ap.), informace ze systémů přednostní jízdy. Jsou zde použity nadefinované signální skupiny.

Uživatel bude mít možnost přidat nový detektor, upravit nebo smazat stávající detektory.

3.6.6 Mezičasy

Mezičasy jsou důležité pro definování časů mezi jednotlivými signálními skupinami. Musí být zajištěna symetrie mezi skupinami.

3.6.7 Signální plány

Signální plány slouží pro definování signálního obrazu křižovatky sekundu po sekundě. Jsou tedy definovány posloupnosti a doby trvání jednotlivých signálních skupin.

Uživatel bude mít možnost definovat nový plán, editovat nebo mazat stávající plán. Dále bude mít možnost kopírovat definice mezi jednotlivými plány a exportovat plán do textového souboru.

3.6.8 Fáze ručního řízení

Jedná se o fáze, které jsou určeny pro acyklické ovládání řadiče z ručního řízení, pro spouštění tras pro vozidla s preferencí (hasiči) přes ústřednu nebo hardwarový spouštěč. Některé fáze mají

předdefinovaný význam. Dále bude implementován algoritmus pro hlídání případných kolizí ve skupinách, ze kterých jsou definovány fáze. Používají se nadefinované signální skupiny.

Uživatel bude mít možnost definovat nové fáze, editovat a mazat stávající.

3.6.9 Rozvrhy

Jsou určeny k lokálnímu automatickému ovládání řadiče v závislosti na datu a čase. Umožňují zapínat a vypínat řadič (do kmitavé žluté nebo kmitavé žluté z vedlejšího směru), měnit číslo signálního plánu, odepínat a připínat jednotlivá záhlaví (sekce) řadiče.

V základním stavu je stav řadiče vždy určen dnem v týdnu, ten určí denní rozvrh, kde se podle času rozhodne o konkrétní akci.

Výjimkou jsou pak svátky, kdy je možné pro den svátku spustit jiný denní rozvrh, než by odpovídalo dni v týdnu.

Další výjimkou jsou speciální periody, které umožní pro vícedenní časové období nastavit jiný týdenní rozvrh.

Uživatel bude mít možnost spravovat denní a týdenní rozvrhy, svátky a specifické periody. Dále má možnost importu a exportu svátků ze souboru.

3.6.10 Výstupy

Pro možnost ovládání výstupů (kromě chodecké signalizace „Čekejte“ řešené jako výstup u detektoru) je nutné definovat výstupy. Ovládání výstupů lze realizovat pouze ve volně programované části v jazyce C.

Uživatel bude mít možnost definovat nové výstupy, editovat a mazat stávající.

3.6.11 MHD

Pomocí tohoto formuláře budou definovány přihlašovací body a preference městské hromadné dopravy. Jsou zde použity signální skupiny.

Uživatel bude mít možnost definovat nový přístupový bod, editovat nebo mazat stávající bod. Dále má možnost definovat nové preference, editovat a mazat stávající.

3.6.12 Tisk

Aplikace umožní uživateli tisk vybraných formulářů samostatně nebo provést souhrnný tisk všech informací dohromady. Tisky slouží jako doklad pro odběratele řadiče SSZ.

3.6.13 Ukládání definice řadiče SSZ

Pro uložení definic řadiče SSZ nesmí být použit žádný databázový server. Uložení musí být provedeno v souboru ve formátu XML. Vzhledem k rozsáhlosti struktury a neúspornosti ukládání

dat ve formátu XML je požadováno ukládání dat co neúsporněji. Úspory lze dosáhnout neukládáním nevyplněných elementů nebo údajů s implicitními hodnotami.

3.6.14 Zabezpečení XML proti modifikaci

Každý XML soubor musí být chráněn proti přepisu jinými prostředky než aplikací CROSS PTC. Takový zásah musí zabránit dalšímu použití XML soboru odmítnutím aplikace takový soubor otevřít. Musí ale existovat možnost zásahem dodavatele XML soubor zase označit za platný. Vše může být v XML volně čitelné, kromě hesla k řadiči SSZ.

4 Návrh implementace

Tato část textu se zabývá popisem rozložení implementace. Aplikace bude rozdělena do dvou základních vrstev. Jedná se o prezentační a datovou vrstvu.

4.1 Prezentační vrstva

Prezentační vrstva má za úkol zobrazovat požadovaná data systému uživatelům a poskytovat jim zároveň přístup k jednotlivým funkcím systému. Jednotlivé formuláře musí být navrženy tak, aby ovládání bylo co nejvíce intuitivní a nezdržovaly uživatele jeho pochopením.

4.1.1 Windows formuláře

Windows formulář je jedno okno operačního systému, které obsahuje libovolné množství dalších vizuálních komponent. Celá aplikace se většinou skládá z libovolného počtu těchto formulářů. Záleží na rozsahu a přehlednosti aplikace. Vizuální komponenty můžou být standardní od společnosti Microsoft, vytvořené vlastní nebo koupené od třetích stran.

4.1.2 Kód na pozadí

Ke každému formuláři existuje přidružený soubor, do kterého programátor vytváří obslužný kód pro daný formulář. V první řadě se jedná o kódy programu starající se o výpis hodnot do vizuálních komponent. Dále se může jednat o volání jiných funkčních knihoven nebo vlastní funkční kód, který vytváří samotnou funkčnost aplikace.

4.2 Datová vrstva

Datová vrstva se bude starat o propojení aplikace s XML souborem jako úložištěm dat. Dále bude zajišťovat zabezpečení přístupu k datům.

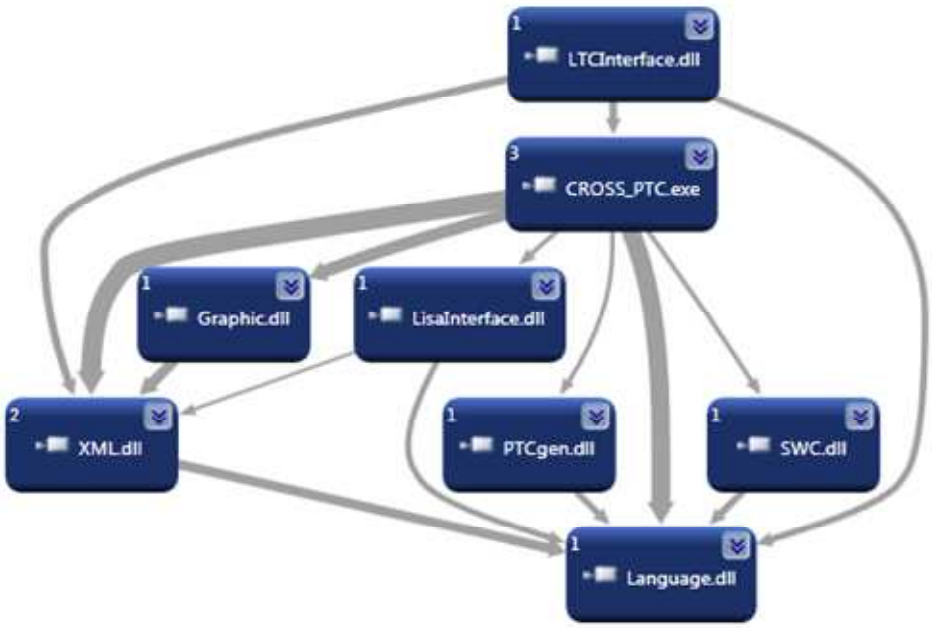
V každém formuláři aplikace se bude pracovat s XML souborem, je výhodné toto zpracování od samotné aplikace oddělit do samotné knihovny funkcí XML.dll, která bude podle potřeby volána. Budou zde implementovány funkce pro zpracování dat pro každý formulář. Pomocí metod se budou předávat informace pro uložení a zpětná data pro vizuální komponenty pro zobrazení uživateli.

5 Implementace

V této kapitole budou popsány principy implementace jednotlivých částí aplikace. Vybral jsem implementačně a funkčně zajímavé části. Pro znázornění architektury aplikace je použit diagram komponent popsaný v sekci 5.1.

5.1 Diagram komponent

Diagram komponent zobrazuje fyzické rozložení jednotlivých softwarových komponent použitých v systému. Může se jednat o vlastní komponenty nebo komponenty zakoupené od třetí strany. V této aplikaci jsou požity jak vlastní, tak i cizí komponenty naprogramované v odlišném programovacím jazyce.



Obrázek 3: Diagram komponent

- *CROSS_PTC.exe* – jedná se o samotnou aplikaci
- *Language.dll* – obstarává načtení souboru s překlady a práci s nimi
- *LisaInterface.dll* – provádí import definice z aplikace LISA+
- *XML.dll* – provádí veškeré práce s XML souborem, od načtení přes úpravy až po samotné uložení
- *Graphic.dll* – vytváří grafické znázornění signálního plánu
- *SWC.dll* – generuje a odesílá soubory do řadiče SSZ
- *LTCInterface.dll* – vytváří propojení mezi aplikacemi CROSS PTC a CROSS LTC
- *PTCgen.dll* – generuje binární soubory pro upload

5.2 Tisk

V této sekci bude popsán způsob implementace tisku definic řadiče. Pro tento tisk výstupů z jednotlivých formulářů byly použity standardní komponenty .NET Frameworku. Do každého formuláře, kde je požadován tisk, byly vloženy do menu tři položky, a to nastavení vzhledu, náhled a tisk. Základní komponentou je *PrintDocument*, přes kterou jsou spojeny další komponenty jako *PrintDialog* (Tisk), *PageSetupDialog* (nastavení vzhledu) a *PrintPreviewDialog* (náhled).

Pro tisk se požívá jmenný prostor *System.Drawing.Printing*, kde je základní třídou *PrintDocument* reprezentující objekt odesílaný do tiskárny. Celý proces tisku začíná vytvořením instance ze třídy *PrintDocument* a zavoláním metody *Print*. Následně je vyvolána událost *PrintPage* s parametrem *PrintPageEventArgs*, který nese informace o nastavení stránky, grafickém objektu typu *Graphic* a zda obsahuje další strany. Do grafického objektu se vykreslí požadované informace pomocí metod v něm implementovaných pro vykreslování textu a grafických útvarů. Dokud tato obslužná metoda tisku bude vracet vlastnost *HasMorePages* na hodnotě *True*, bude tato metoda neustále volána, dokud se nevrátí hodnota *False* signalizující konec dokumentu.

Náhled tisku se dá zobrazit pomocí komponenty *PrintPreviewDialog*. Objekt *PrintDocument* se propojí s objektem *PrintPreviewDialog* pomocí vlastnosti *Document*. Zobrazení se provede zavoláním metody *ShowDialog*, která opět vyvolá již definovanou událost *PrintPage*. Kreslící algoritmus v události *PrintPage* musí být jak pro náhled, tak i pro tisk shodný. Platí tedy, že náhled i vytištěný dokument na papíře jsou graficky shodné.

Nastavení vzhledu stránky se provádí pomocí komponenty *PageSetupDialog*. Opět je propojena s objektem *PrintDocument* pomocí vlastní *Document*. Uživatel zde může nastavit typ stránky a okraje pro tisk.

Funkce tisku je implementována v samostatné třídě s názvem *Printer*. Zavoláním tisku se vytvoří instance z této třídy. Jako první parametr se předává objekt dokumentu, na který se bude tisknout, dále se předávají hodnoty, zda má být text vystředěn, vytisknuta hlavička stránky, text hlavičky stránky, font písma tisku, barvu písma, zda se mají stránky číslovat, typ tisku tabulky a data pro tisk všech požadovaných informací. Vytvoření tohoto objektu je ukázáno na následujícím obrázku.

```
private void bPrint_Click(object sender, EventArgs e)
{
    DialogResult res = printDialog.ShowDialog();
    if (res == DialogResult.OK)
    {
        printer = new Printer(printDocument, false, true, this.lang.GetText("Print.Sections"),
            new Font("Tahoma", 10, FontStyle.Bold, GraphicsUnit.Point), Color.Black,
            false, 0, this.xml);
        printDocument.Print();
    }
}
```

Obrázek 4: Vytvoření instance tisku a následné zavolání tisku

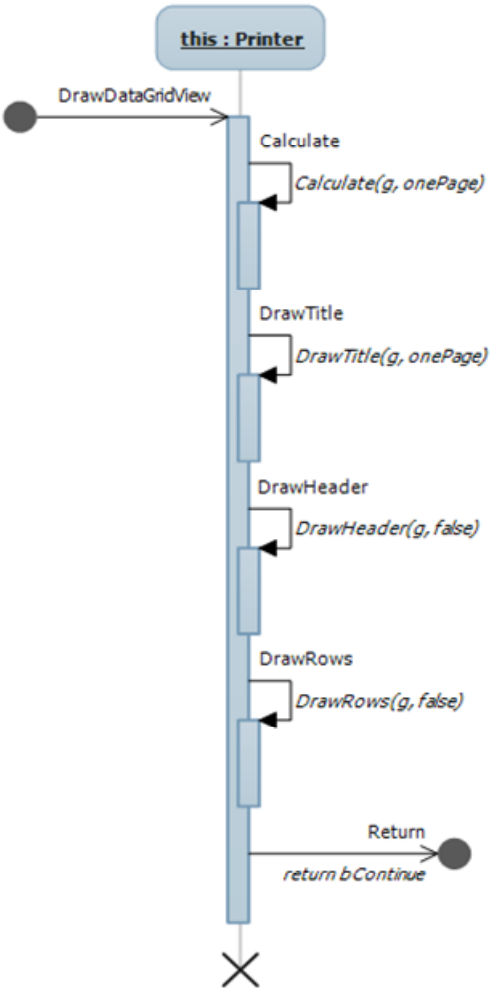
Jak již bylo psáno výše, zavoláním metody *Print* se vyvolá událost *PrintPage* zobrazená na následujícím obrázku. Zde je volána metoda pro vykreslení požadovaných informací na stránku.

```
private void printDocument_PrintPage(object sender, PrintPageEventArgs e)
{
    bool more = printer.DrawDataGridView(e.Graphics, dgvSection, columns, false);
    if (more == true)
        e.HasMorePages = true;
}
```

Obrázek 5: Událost PrintPage

S tiskem nám vzniká problém s posouváním dokumentu, který nám tiskárny neumožňují, místo toho tisknou novou stránku. Je tedy nutné najít rozumný způsob jak dokument rozdělit na jednotlivé stránky a vypočítat, jaká část dokumentu se na stránku vejde.

Do proměnné *more* se ukládá hodnota, zda tištěný dokument má ještě nějakou stránku k vytištění. Pokud je vrácena hodnota *True*, událost je volána dokud se nevrátí hodnota *False* a tisk je u konce. Základní metodou pro tisk tabulek je *DrawDataGridView*. Jako parametr se předává objekt typu *Graphic* pro kreslení grafických objektů na stránku. Dalším parametrem je komponenta s tabulkovým výpisem, dále seznam sloupců, které se z této tabulky mají tisknout a zda dokument bude mít více než jednu stránku.



Obrázek 6: Sekvenční diagram - tisk

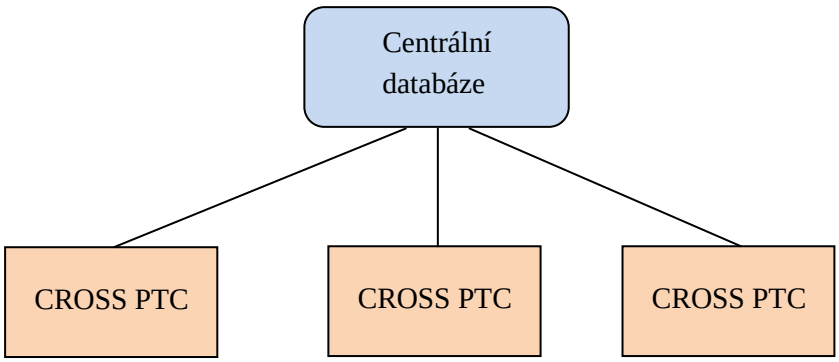
Na obrázku Obrázek 6 je zobrazen sekvenční diagram tisku. Vytištění informací se skládá z několika volání metod. Jako první je volána metoda *Calculate*, která se stará o výpočet výšky jednotlivých řádků a šířky jednotlivých sloupců v tabulce podle zvolené velikosti písma a obsahu. Dále je volána metoda *DrowTitle*, vykreslující čísla stránek a hlavičku stránky s názvem křižovatky. Metoda *DrawHeader* vykresluje hlavičku tabulky. O následné vykreslení řádků tabulky se stará metoda *DrawRows*. Zde byla popsána funkce tisku tabulkové komponenty, ne vždy však pouze tento typ tisku postačoval. Pro tisk plánů, fází, řadiče a mezičasů bylo potřeba vytvořit samostatné metody pro tisk kvůli jejich odlišnostem. V principu ale fungují stejně jako popsaný princip výše [1].

5.3 Zabezpečení

V této části budou popsány různé možnosti řešení ověřování uživatelů pro přístup do aplikace. Dalším bezpečnostním problémem, který bylo nutné vyřešit je ochrana XML souboru proti modifikaci mimo aplikaci CROSS PTC.

5.3.1 Přihlášení do aplikace

Prvním a asi nejpoužívanějším způsobem ověřování uživatelů pro přístup do windows form aplikací je využití centrální databáze s uživateli. Zde by mohly být uloženy i veškeré uživatelské nastavení aplikace. Komunikace se serverem by probíhala pomocí volání webové služby, která by vracela potřebné informace o uživateli a uloženou konfiguraci na serveru. U tohoto řešení je nutné, aby byl počítač, na kterém chce uživatel spustit aplikaci, vždy připojen k internetu. V opačném případě se aplikace vůbec nespustí nebo se spustí jen v nějakém omezeném režimu. Toto řešení má za výhodu snadnou správu uživatelských účtů pro přístup. Architektura toho přístupu je zobrazena na obrázku Obrázek 7.



Obrázek 7: Možný způsob ověřování uživatelů

Dalším způsobem by mohlo být ověření uživatele proti lokální databázi spuštěné na stejném počítači jako je aplikace, nebo lokálním souboru obsahující jméno a heslo pro přístup. V tomto řešení je obtížnější správa a distribuce seznamu uživatelských účtů.

Jelikož je vyvíjená aplikace využívána na počítačích bez stálého připojení k internetu, není tedy možné použít architekturu ověřování uživatele proti centrální databázi uživatelů. Zabezpečení přístupu do aplikace je řešeno pomocí uživatelského jména a hesla. Každá distribuce aplikace obsahuje externí binární soubor se seznamem uživatelů. Přístup je rozdělen do několika rolí, dle kterých se aplikace konfiguruje. Každá role má přístup jen ke specifickým funkcím odpovídající roli. Pokud potřebuje zákazník upravit uživatele, popřípadě přidat dalšího, je nutno zažádat o změnu přihlašovacího souboru. Ten je pak vystaven na místě, kde mají zákazníci využívající aplikaci přístup a mohou si jej stáhnout, popřípadě je soubor distribuován jako aktualizace aplikace.

Veškeré informace o uživatelích jsou binárně uloženy a speciálním šifrovacím algoritmem zašifrovány v souboru AUTH.DAT.

Požadavek na aplikaci byl, že musí pracovat bez nutnosti připojení k internetu a musí být schopna ověřit uživatele a přiřadit jej do role, což bylo tímto řešením splněno.

5.3.2 Zamezení modifikace XML souboru

Velký důraz při vývoji aplikace byl kladen na zabezpečení XML souboru proti modifikaci jinými nástroji, než je aplikace CROSS PTC.

Princip hlídání modifikace souboru je založen na generování určitého podpisového klíče vkládaného do XML souboru. Při každém pokusu o otevření definice provede aplikace kontrolu podpisového klíče. Nejprve byl vyvinut způsob ověřování, který byl ale závislý na počítači, na kterém byl soubor podepsán. Toto vlastnost byla postupem času nežádoucí, protože jsou definice spravovány více uživateli na více počítačích. Současná verze podpisu souboru je nezávislá na počítači uživatele.

Podrobnější informace se nacházejí v interní dokumentaci firmy. Z důvodu zachování bezpečnosti aplikace další podrobné informace nebudou zveřejněny. Podrobný dokument zabývající se zabezpečením XML souboru se nachází v literatuře [6].

5.4 Vizuální komponenty

V aplikaci byly použity jak standardní vizuální komponenty společnosti Microsoft dodávané jako součást vývojového prostředí, tak i vlastní z nich vytvořené. Vlastní komponenty má význam vytvářet pokud je vkládána do aplikace vícekrát.

Každý formulář umožňuje zapnout grafickou funkci Double buffering. Bez této funkce se může objevovat problikávání při vykreslování grafických prvků. Jedná se o funkci, která využívá dvojitou paměť pro vykreslení. Více v literatuře [2].

Po zapnutí této funkce pořád docházelo k problikávání při vykreslování formulářů, hlavně u signálního plánu a tabulkových komponent. Zkoumáním se prokázalo, že povolením vylepšeného vykreslování u formuláře se tato vlastnost nepřenesla na komponenty v něm vložené. Proto byla každá komponenta, která je náročnější na vykreslení přepsána na vlastní komponentu se zapnutou

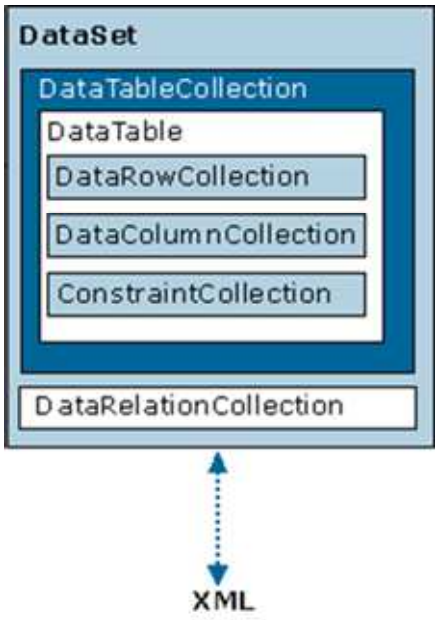
funkcí Double buffering. Na obrázku Obrázek 8 je ukázka kódu vytvoření vlastní komponenty *NewDataGridView* poděděním z původní komponenty *DataGridView*. V konstruktoru jsou pak definovány vlastnosti stylu vykresleni.

```
public class NewDataGridView : DataGridView
{
    public NewDataGridView()
    {
        this.SetStyle(ControlStyles.AllPaintingInWmPaint |
                      ControlStyles.UserPaint |
                      ControlStyles.OptimizedDoubleBuffer, true);
        this.UpdateStyles();
    }
}
```

Obrázek 8: Nové komponenty s vylepšenou grafikou

5.5 *Práce s XML soubory*

Dále bude vysvětlen princip práce s XML souborem v prostředí .NET Framework. Podpora zpracování XML se v .NET frameworku nachází ve jmenném prostoru *System.Xml*. Základní třídou pro několik dalších tříd v tomto jmenném prostoru je *XmlNode*, která reprezentuje jediný uzel dokumentu. Potomkem této třídy je třída *XmlDocument*, která reprezentuje stromovou strukturu XML dokumentu v paměti. Potomkem *XmlDocumentu* je třída *XmlDataDocument*, která je schopna načíst XML nebo relační data do objektu typu *DataSet*. Zajišťuje most mezi technologií ADO.NET a XML dokumentem.



Obrázek 9: Architektura části ADO.NET a XML [15]

Jmenný prostor pro zpracování XML dat samozřejmě obsahuje daleko více tříd. Dalšími důležitými třídami pro práci s XML jsou *XmlReader*, *XmlTextReader*, *XmlWriter*, *XmlTextWriter*. Tyto třídy zajišťují rychlý přístup pro čtení/zápis bez mezipaměti ve formátu datového proudu.

Výhodou technologie ADO.NET je, že nezáleží na původu dat, zda byla načtena z databáze (SQL server nebo Oracle) nebo XML souboru. Základní třídou pro ukládání dat v paměti je třída *DataSet*, která byla navržena jako datový kontejner. Obsahuje sadu datových tabulek *DataTableCollection*. Každá z nich má sadu datových sloupců *DataColumnCollection* a řádků *DataRowCollection*. Datová tabulka může mít definovaný jak primární klíč, tak i omezení cizích klíčů. Do kolekce *Contraints* lze uložit unikátní omezení nebo omezení primárního klíče, který jednoznačně identifikuje záznam. *DataRelationCollection* slouží pro uložení relací mezí tabulkami jako je například závislost jedné tabulky na druhé.

Práce s XML souborem je odděleno od samotné aplikace. Nachází se v DLL knihovně s názvem XML.dll. Základem celého zpracování XML souboru je jeho převedení na databázová data technologie ADO.NET uložená v objektu typu *DataSet*. Seznam tabulek a vlastností v nich se vytvoří podle XSD schématu. V následujícím kusu kódu je ukázán způsob načtení XML dokumentu do objektu typu *DataSet* v paměti aplikace.

```
XmlDataDocument xmlDataDocument = new XmlDataDocument();//vytvoření instance
xmlDataDocument.Schemas.Add(null, xsdFilePath);//přiřazení XSD schématu

DataSet dataSet = xmlDataDocument.DataSet;//propojení DataSetu s XmlDataDocumentem
dataSet.ReadXmlSchema(xsdFilePath);//načtení XSD schématu do Datasetu

xmlDataDocument.Load(path);//nacteni xml souboru
```

Obrázek 10: Ukázka načtení XML souboru

Nad vytvořeným *DataSet* jsou prováděny veškeré operace s daty. Pří změně hodnot v *DataSet* se změna automaticky projeví i v objektu *XmlDataDocument*, pomocí kterého se provádí kromě načtení i zpětné uložení definice do souboru. Změny se samozřejmě projeví i obráceně. Oba objekty tedy pracují nad jedním souborem údajů. Propojení mezi objektem *DataSet* a *XmlDataDocument* vytváří velmi flexibilní přístup jak ke všem službám okolo třídy *DataSet*, tak i okolo zpracování XML dokumentu (Xml tranformace). Výhodou je dostupnost obou přístupů a není nutno jeden z nich vybírat [10].

5.6 Předávání informací se SW systémem LTC

Jak bylo psáno v kapitole 2.12 aplikace LTC je určena k monitorování řadičů SSZ. Je napsána v odlišném programovacím jazyce DELPHI, bylo tedy třeba připravit rozhraní, které umožní vzájemné volání přes DLL knihovnu.

Vše je řešeno v knihovně LTCInterface.dll, která obsahuje metody pro volání jednotlivých částí aplikace PTC bez vzájemné návaznosti. Při tomto volání je řešeno jiné ověření uživatele a výběru města, křižovatky a verze. Z pohledu PTC je tento krok vynechán a potřebné informace se musely aplikaci „podsunout“ z aplikace LTC, kde již k ověření uživatele a k výběru křižovatky došlo.

5.7 Práce s formuláři

5.7.1 Lokalizace a globalizace

Pokud je aplikace využívána mezinárodně je potřeba provést její lokalizaci a globalizaci.

- Globalizace znamená, že vytvořená aplikace funguje správně v operačním systému v libovolném jazyce a lokálním nastavením.
- Lokalizace znamená, že aplikace má oddělené zdroje od samotného kódu a jsou závislé na zvoleném jazyce. Takto vytvořenou aplikaci lze pak snadno doplnit o libovolný počet dalších jazykových překladů.

Lokalizace

Lokalizace windows form aplikací je podporována ve vývojovém prostředí Microsoft Visual Studio. Pokud chcete formulář lokalizovat, stačí v grafickém návrháři nastavit hodnotu *Lokalizable* ve vlastnosti formuláře na hodnotu *True*. Touto změnou se povolí automatické ukládání vlastností do resources (.resx) souborů. Dále se ve vlastnostech formuláře nastaví vlastnost jakyku (*Language)* na požadovaný jazyk, pro který se vytvoří samostatný soubor zdroje resx. Tyto soubory jsou pak zkompilovány do satellite assemblies. V souboru formulář.resx je uloženo výchozí nastavení. Pokud přidáme další překlad, například angličtinu, vygeneruje se soubor z názvem formulář.en.resx, kde jsou uloženy všechny texty, když se aplikace vyvíjela ve zvoleném jazyce [7].

Z důvodů požadavků jsem pro lokalizaci zvolil vlastní způsob lokalizace, která je řešena pomocí externího CSV souboru. V něm jsou uloženy všechny požadované jazykové mutace dohromady. Princip je plně dynamický a nezávislý na kompilaci aplikace, což byl jeden z požadavků zadavatele. Dalším z požadavků byla snadná editace textů. Tento soubor lze editovat buď pomocí poznámkového bloku, který je součástí operačního systému Microsoft Windows, nebo pomocí aplikace Microsoft Excel. Není tedy potřeba instalace jiných programů, než těch, které uživatel může běžně nalézt ve svém počítači.

První řádek souboru slouží jako hlavička obsahující definici použitých jazyků překladu aplikace oddělených středníkem. Tato informace je pro aplikaci výchozí a podle ní se udává počet jazykových překladů v aplikaci a načítání jednotlivých záznamů překladu v daném jazyce. První záznam „Component name“ označuje, že nejprve musí být vždy zadán název komponenty, ke které se má daný text přiřadit. Za středníkem pak následují názvy jednotlivých jazyků. Na dalších řádcích souboru jsou samotné překlady ke komponentám ve formuláři, překlady hlášek apod.

```
Component name;Czech;English;German
Czech;Česky;Czech;Tschechisch
English;Anglicky;English;Englisch
German;Německy;German;Deutsch
Login.text;Přihlásit;Login;Anmeldung
Login.lUserName;Přihlašovací jméno;Username;Benutzername
```

Obrázek 11: Ukázka lokalizačního souboru

Od počátku vývoje byla lokalizace součástí aplikace CROSS PTC jako samostatná třída. Načtení souboru s překlady probíhalo pouze jednou a při spouštění aplikace. Postupem času bylo potřeba lokalizovat další dynamické knihovny (DLL) nezávisle na vyvíjené aplikaci, proto byla lokalizace přepracována jako samostatná DLL knihovna starající se o správu všech překladů definovaných v souboru *Language.csv*. Knihovna obsahuje třídu s názvem *GetLang*, ve které se při vytváření objektu načítá jazykový soubor. Pro uložení překladů je použita třída *Dictionary* (slovník), která je součástí .NET frameworku. V souboru Setting.ini je uloženo číslo jazykové lokalizace, kterou si uživatel nastavil v konfiguračním formuláři aplikace. Podle tohoto identifikátoru jsou překlady textů ukládány do slovníku. Dále tato třída obsahuje metodu *GetText*, která vrací překlad ze slovníku. Jako parametr se předává identifikační název komponenty, pro kterou je překlad určen. Dále obsahuje metodu *SetLanguageText*, která nastavuje překlad přímo komponentě. Jako parametr se předává přímo komponenta a název formuláře, ve které je komponenta vložena. Jelikož aplikace byla postupem času použita jako modul pro jinou aplikaci a formuláře mohou být zobrazovány v libovolném pořadí, je potřeba při každém otevření formuláře načíst jazykový soubor a provést lokalizaci.

```
for (int i = 0; i < this.Controls.Count; i++)
{
    this.lang.SetLanguageText(Controls[i], "Detector");
}
```

Obrázek 12: Ukázka volání lokalizace

Globalizace

.NET framework pro globalizaci využívá jmenný prostor *System.Globalization*, který obsahuje třídy všech kultur a regionů, které podporují různé formáty data a čísel. Základní třídou je *CultureInfo* umožňující zjištění základního systémového nastavení. Pomocí statické metody *CultureInfo.CurrentCulture* lze získat aktuální nastavení. Výchozí nastavení pro jazyk a zemi lze získat pomocí kódu definující specifikace RFC. Tyto kódy mají formát xx-YY, kde xx představuje zemi a YY určuje nastavení konkrétní země. Kód pro angličtinu je en, pro nastavení americké angličtiny se tedy použije kód en-US a při nastavení britské angličtiny se požije kód en-GB [1].

5.7.2 Současné otevření více oken

Při definování křižovatky je občas potřeba otevřít více oken z hlavního okna aplikace současně. Jelikož informace definované v jednom formuláři mohou ovlivňovat informace definované v jiných

formulářích, bylo nutné programově zabezpečit současné otevírání těchto oken a následné nekonzistentní uložení.

Základní omezením všechno formulářů je, že jeden formulář může být otevřen současně pouze jednou. Pokud nastane situace, kdy je otevřen jeden formulář a uživatel chce otevřít další formulář, který je ale závislý na prvně otevřeném formuláři, tak tyto další formuláře jsou otevírány pouze pro čtení. V následující tabulce (Tabulka **1**) jsou zobrazeny závislosti mezi jednotlivými částmi aplikace při editaci.

Editovaný formulář	**Závislé formuláře**												
	Řadič	**GSM**	**Sekce**	**Skupiny**	**Napájení**	**Detektor**	**Mezičasy**	**Kolize**	**Plány**	**Fáze**	**Rozvrhy**	**Výstupy**	**MHD**
Řadič													
GSM													
Sekce				X									
Skupiny			X		X	X			X	X			
Napájení				X									
Detektory				X									
Mezičasy				X				X	X	X			
Kolize				X			X		X	X			
Plány				X			X						
Fáze				X			X						
Rozvrhy									X				
Výstupy													
MHD				X									

Tabulka 1: Vzájemné ovlivňování formulářů

V prvním sloupci je seznam všech základních formulářů aplikace. Pokud je v některém sloupci křížek, znamená to, že při otevírání formuláře pro editaci nesmí být formulář s tímto křížkem otevřen. Pouze formuláře pro definici řadiče, GSM a výstupů mohou být editovány současně, bez vzájemného omezení.

5.8 Logování chyb

Podrobné logování chyb vyskytlých se v aplikaci je užitečné pro snazší a rychlejší odstranění chyby z aplikace programátory. Jedná se o framework pro .NET Runtime. Log4net je schopný chyby ukládat do různých datových zdrojů, jako jsou různé databáze, textové soubory nebo posílat na vzdálené servery. V současné verzi zatím není tento nástroj implementován. Je s ním počítáno v další verzi.

5.9 Kompilace pro LINUX

Jelikož řadič SZZ pracuje pod operačním systémem Linux je potřeba provést kompilaci volné programovatelné části napsané v jazyce C.

Řešení:

- GCC cross compiler pro ARM
- část cygwin pro prostředí, ve kterém by se dal spouštět
- editace spouštěna z aplikace
- kompilace spouštěna z aplikace s výpisem výsledku
- vytváření linkovatelného souboru *.o, který se přenese do řadiče jako další z datových souborů
- výchozí zdrojový soubor bude mít pořád stejnou hlavičku funkcí, tím se zajistí možnost jej kdykoliv přilinkovat k firmware řadiče

5.10 Import z LISA+

Import z aplikace LISA+ je implementován v knihovně LisaInterface.dll. Aplikace prochází požadované elementy v XML souboru a převádí data na formát aplikace CROSS PTC. Během importu se podrobně zaznamenává vše, co bylo do definice řadiče přidáno. Po dokončení je uživateli zobrazena zpráva o průběhu importu.

5.11 Generování binárních souborů

Generování souborů je navrženo a realizováno tak, aby objem datových souborů byl objemově co nejúspornější. Všechny potřebné nástroje jsou implementovány v knihovně PTCgen.dll. Přenos se provádí často po zařízeních s rychlostí 9600 bit/s s poměrně velkou odezvou (např. GSM datová komunikace). Při realizaci se ale nemohla použít komprimace, protože na straně příjemce nelze dekomprimaci zajistit. V generovaných souborech se nepřenáší kompletní datová struktura dimenzovaná na maximální počty objektů a maximální počet vyplněných údajů.

Datový soubor je opatřen identifikací:

- kdy vznikl
- kdo jej vytvořil
- jakou verzí SW

Uložená hesla v souboru jsou nečitelná. Generování je zpětně kompatibilní. Software v řadiči pracuje s libovolnou verzí datového souboru. Ve skutečnosti se nejedná o jeden soubor, ale sadu částí, na které je datový soubor rozdělen. Definice těchto částí je neveřejnou součástí firemní dokumentace.

V režimu offline se předávání datových souborů provádí přes externí USB disk pomocí složky s pevně daným jménem.

5.12 Kreslení signálního plánu

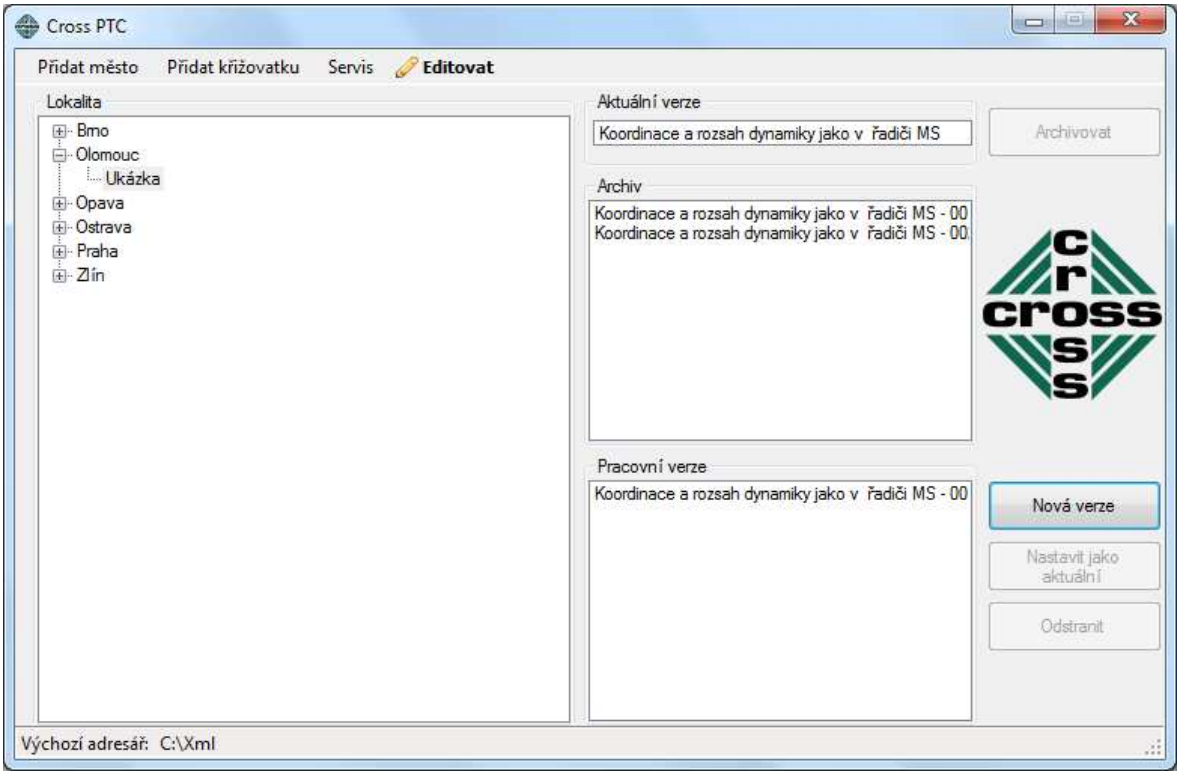
Vykreslování signálních plánů je implementováno v knihovně Graphic.dll. Jsou zde implementovány metody pro vykreslení jednotlivých znaků obsažených v signálním plánu. Pro vykreslení jsou vytvořeny dvě základní metody, první z nich vykresluje skupiny na levé straně signálního plánu. Druhá metoda se stará o vykreslení časové, fázové osy a samotné volání vykreslení jednotlivých znaků. Vykreslení je plně dynamické, signální plán pro určité situace může být posunut od počátku o libovolný počet časových jednotek.

6 Ukázky aplikace

V této kapitole budou ukázány a popsány obrázky funkčnosti z vyvinuté aplikace.

6.1.1 Správa definic řadičů

Úvodní obrazovka aplikace pro správu definic řadičů a jejich verzí popisované v kapitole 3.6.2



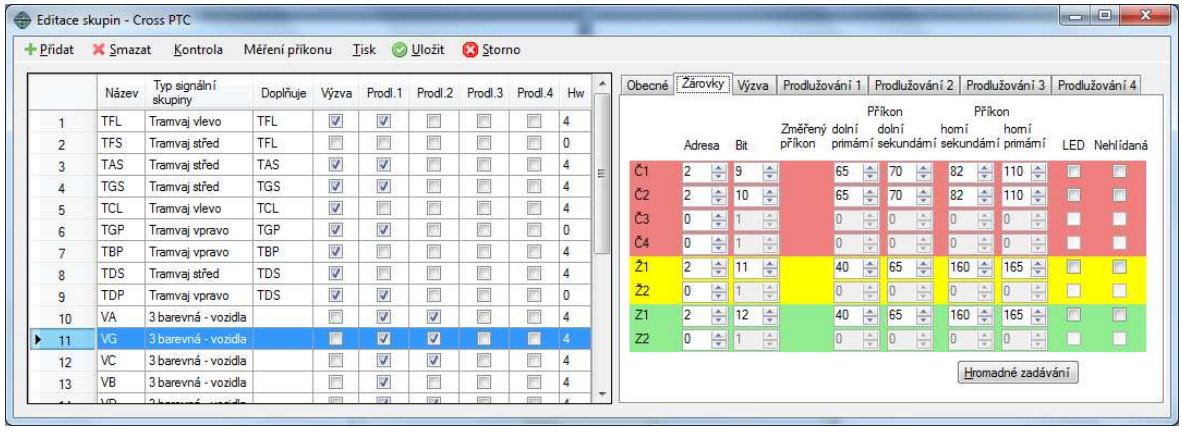
Obrázek 13: Hlavní okno aplikace

Na obrázku Obrázek 13 je zobrazeno hlavní okno aplikace. Zde uživatel může zakládat nové města a nové názvy křižovatek v nich. Dále vytvářet jednotlivé verze definic, jako je aktuální, archivní a pracovní. Kouždou tatko vytvořenou definici lze editovat nebo prohlížet.

6.1.1 Editace signálních skupin

Signální skupiny byly popsány v kapitole 3.6.3. Kromě definování jednotlivých signálních skupin (Obrázek 14: Editace signálních skupin) má uživatel možnost spustit kontrolu, zda při definici nedošlo ke kolizím v signálních skupinách (Obrázek 15).

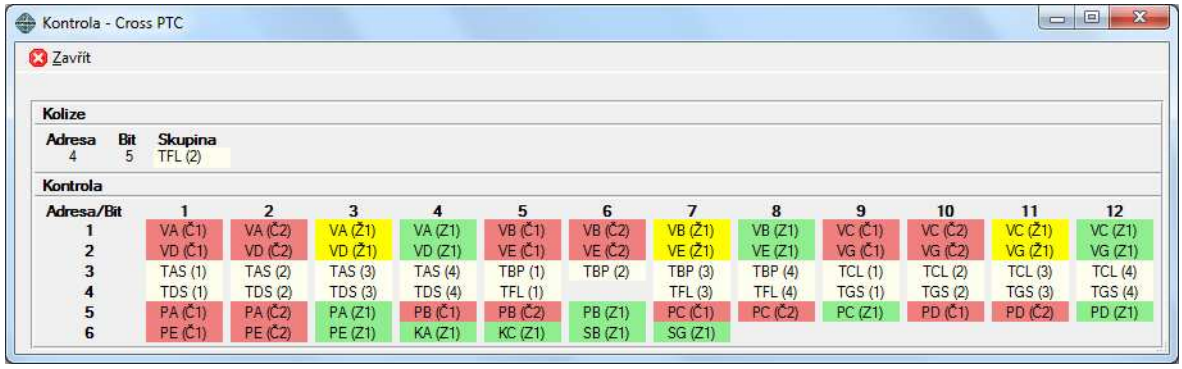
- Formulář pro definování signálních skupin.



Obrázek 14: Editace signálních skupin

Ke každé signální skupině se definují příkony žárovek, adresy, bity a mnoho jiných doplňujících vlastností.

- Následná kontrola kolizí

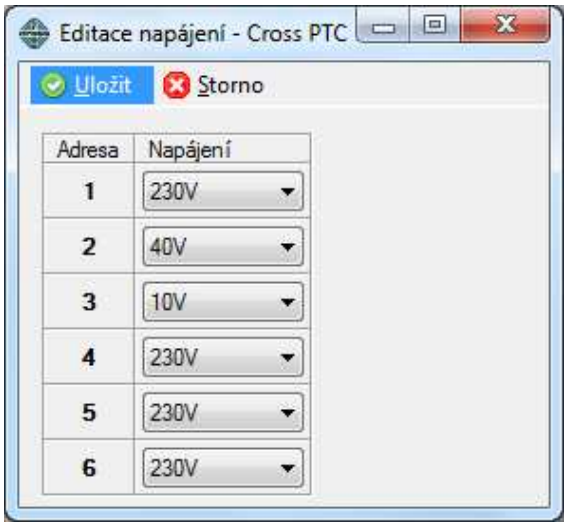


Obrázek 15: Kontrola skupin

Uživatel zde může vizuálně zkontrolovat definice adres a bitů u skupin.

6.1.2 Editace napájení

- Definování napájení na jednotlivých adresách

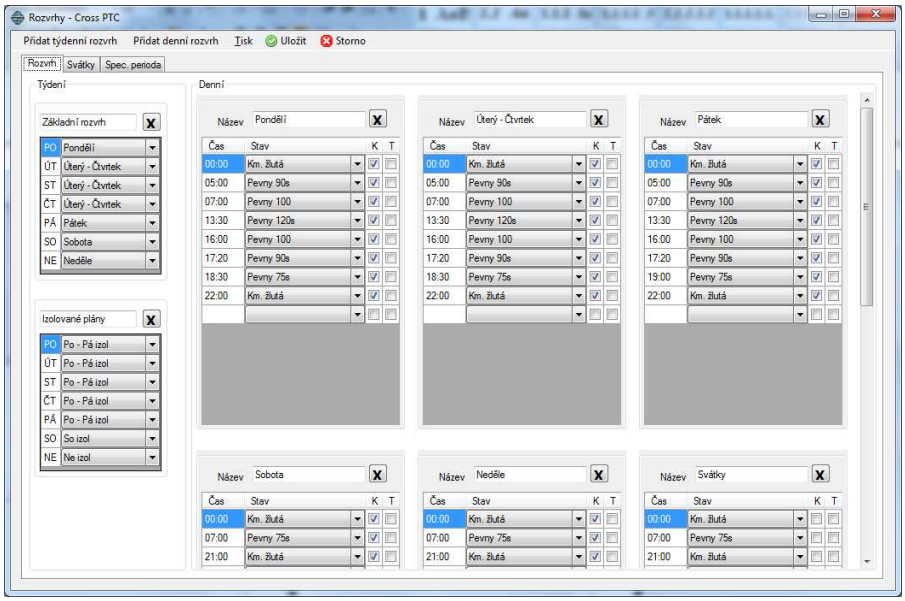


Obrázek 16: Editace napájení

Pro každou definovanou adresu v signálních skupinách odpovídá zvolené napájecí napětí pro světelný zdroj.

6.1.3 Editace rozvrhů

- Formulář sloužící pro definici použití signálních plánů podle aktuálního data a času.

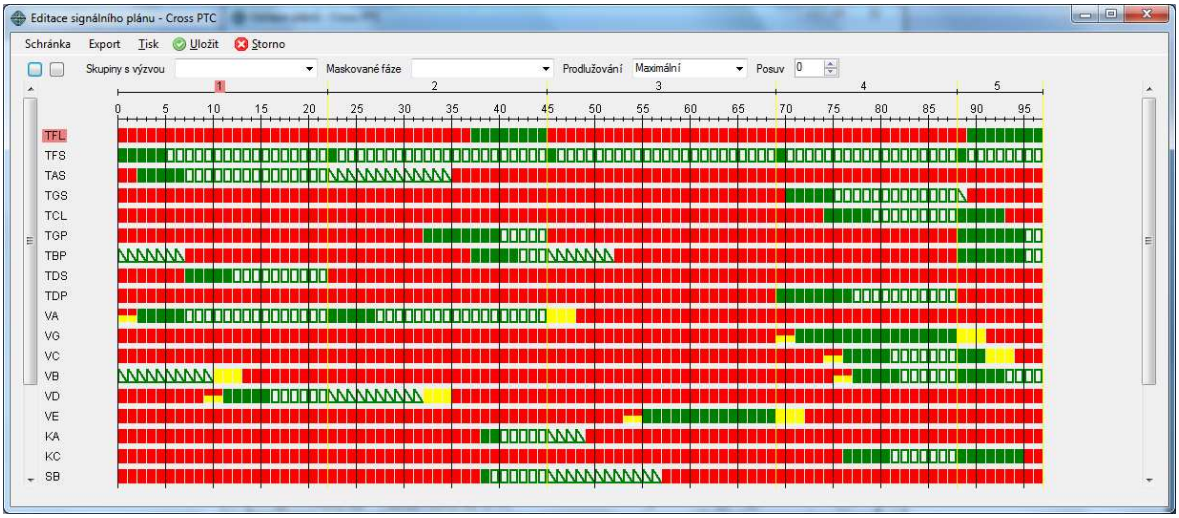


Obrázek 17: Editace rozvrhů

Obrázek Obrázek 17 zobrazuje definici rozvržení signálních plánů podle času v jednotlivých dnech týdne.

6.1.4 Editace signálního plánu

- Definice signálního plánu

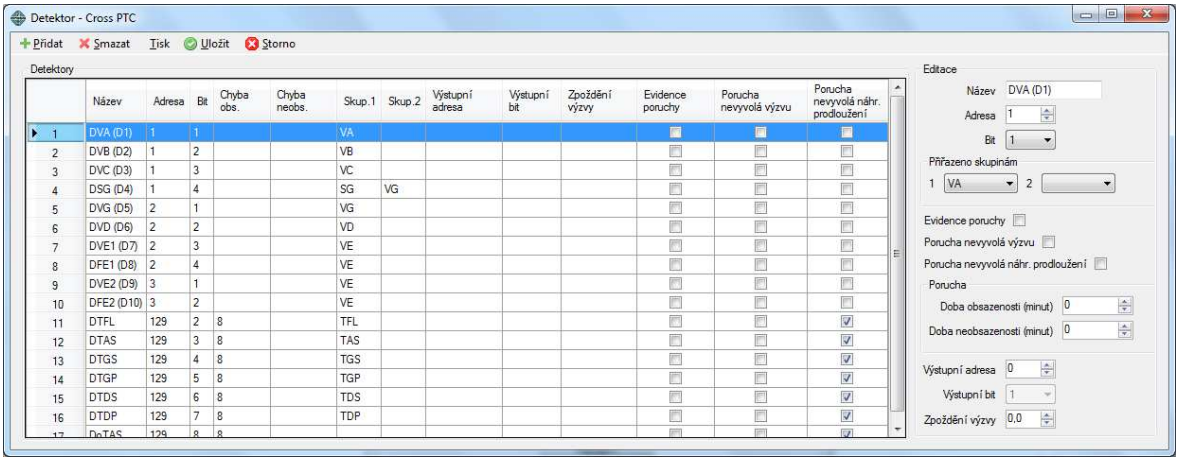


Obrázek 18: Editace signálního plánu

Na obrázku Obrázek 18 je zobrazena definice jednoho signálního plánu (kapitola 2.6). Popisuje chování jednotlivých signálních skupin v čase.

6.1.5 Detektory

- Správa detektorů



Obrázek 19: Detektory

Detektory (kapitola 2.8) slouží pro detekci vozidel přijíždějících ke křižovatce. Na obrázku Obrázek 19: Detektory jsou zobrazeny definice všech detektorů potřebných pro inteligentní řízení provozu.

6.2 Technologie

V této části textu budou popsány jednotlivé technologie použité při implementaci aplikace.

6.2.1 Microsoft Visual Studio

Microsoft Visual Studio je plně integrované vývojové prostředí od firmy Microsoft. Může být použito pro vývoj jak konzolových, tak i grafických aplikací typu Windows forms nebo Web forms s podporou webových služeb. Poslední oficiální verzí je Visual Studio 2008, ale můžete se setkat i se staršími verzemi, jako je například Visual Studio 2005. Při vzniku tohoto textu byla uvolněna první verze Visual Studio 2010.

Součástí Visual Studia je editor kódu podporující IntelliSense, nástroje pro ladění jak na úrovni zdrojového kódu, tak na úrovni strojového kódu. Dalším integrovaným nástrojem je grafický návrhář pro tvorbu grafické aplikace, webové aplikace, třídního diagramu a schématu databáze. Pomocí zásuvných modulů (plug - in) ho lze rozšířit o další funkčnosti.

Ve Visual Studiu lze vytvářet aplikace v různých programovacích jazycích jako je Visual Basic, Visual C#, Visual J#, Visual C++ [5].

6.2.2 NET Framework 2.0

.NET je zastřešující název pro soubor technologií v softwarových produktech, který tvoří celou platformu, jež je dostupná pro Web, Windows i Pocket PC. Základní komponentou je *Microsoft .NET Framework*, prostředí potřebné pro běh aplikací. Nabízí jak spouštěcí rozhraní, tak i potřebné knihovny. Microsoft .NET Framework je nerozšířenější platforma pro osobní počítače s operačním systémem Microsoft Windows. Není to vývojové prostředí, ale „základ“, na kterém je postaveno psaní vlastních aplikací. Zapouzdřuje spoustu funkcí, které můžeme libovolně použít. Ve třídách jsou zapouzdřeny systémové funkce, funkce pro přístup k databázím, komunikace s okolním světem a další jiné funkce. S využitím Frameworku se dají napsat WindowsForm aplikace, webové aplikace, webové služby a aplikace pro malá zařízení.

Verze .NET Frameworku

- .NET Framework 1.0 - historicky první verze. Nové vývojové prostředí Microsoft Visual Studio .NET
- .NET Framework 1.1 - tato verze nepřinášela koncepční změny, ale spíše aktualizace.
- .NET Framework 2.0 - zcela nová a v mnohých ohledech převratná verze .NET Frameworku. Přináší opět nové Studio: Microsoft Visual Studio 2005. Stejně jako v předchozích verzích platí, že verze VS je pevně spjata s verzí frameworku.
- .NET Framework 3.0 - nová verze je sada rozšiřujících knihoven pro .NET 2.0. Aplikace pro V 3.0 se vyvíjejí v předchozí generaci vývojových nástrojů (VS 2005 atd.).
- .NET Framework 3.5 - jedná se opět o rozšíření verze 2.0, ale velmi zásadní. Pro ASP.NET přináší hromadu nových serverových ovládacích prvků a nové vývojové prostředí Microsoft Visual Studio 2008

- .Net Framework 4 – Nejedná se o nádstavbu předchozích verzí, přináší nové vývojové prostředí Visual Studio 2010. Zatím vydána verze Release Candidate

Struktura .NET Frameworku je nastíněna na obrázku 1. Na nejnižší úrovni se nachází *CLR* (Common Language Runtime) realizující základní infrastrukturu, nad kterou je vývojový rámec vybudován. V prostředí .NET jsou zdrojové soubory libovolného programovacího jazyka zkompilovány do intermediárního jazyka (nazvaného *MSIL* – Microsoft Intermediate Language). V případě, že má být taková aplikace spuštěna, systém detekuje, že jde o aplikaci v MSIL a spustí Just-In-Time kompilátor. Ten vygeneruje skutečné instrukce cílové platformy. Jedním z hlavních cílů při vývoji .NET je podpora různých programovacích jazyků. Důležitým prvkem CLR je podpora společného typového systému (Common Type Systém – CTS). Další vlastnosti které CLR také implementuje je mechanismus zajišťující typovou bezpečnost a automatický management paměti.

Nad CLR se nachází několik hierarchicky umístěných knihoven, které jsou rozděleny do jmenných prostorů. Základem je knihovna nazvaná Base Class Library.

Nad ní je knihovna pro přístup k datům a práci s XML soubory.

Další vrstvou je sada knihoven usnadňující práci s uživatelským rozhraním. Je rozdělena do dvou skupin: pro usnadnění vytváření webových aplikací a pro vytváření klasických Windows Forms aplikací.

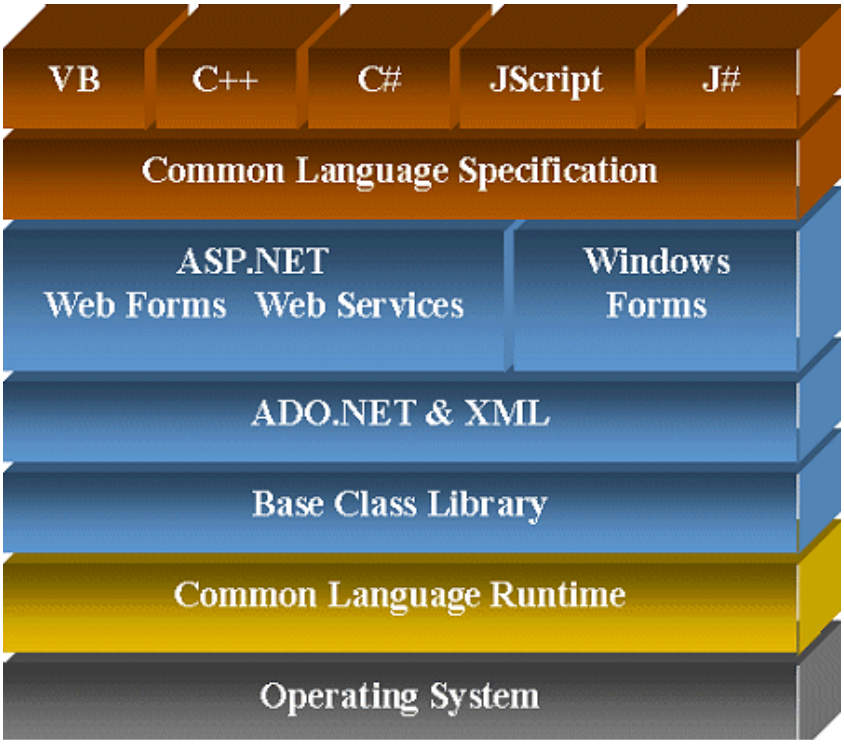
Poslední vrstvu tvoří nelimitovaná množina programovacích jazyků. Jejich základní vlastnosti definuje CLS – Common Language Specification. V současné době je firmou Microsoft podporováno pět jazyků Visual Basic, C++, C#, Jscript a J#. Tato množina není ale uzavřena a jakýkoliv výrobce ji může rozšířit [5].

6.2.3 C#

C# je vysoko úrovňový objektově orientovaný programovací jazyk vyvinutý zároveň s platformou .NET Framework firmou Microsoft, později schválený standardizačními komisemi ECMA a ISO. Microsoft založil C# na jazycích C++ a Java (je tedy nepřímým potomkem jazyka Java, ze kterého čerpá syntaxi). Je to moderní objektově orientovaný programovací jazyk podporující zapouzdření, dědičnost a polymorfismus.

C# lze využít k tvorbě databázových programů, webových aplikací a stránek, webových služeb, formulářových aplikací ve Windows a softwaru pro mobilní zařízení (PDA a mobilní telefony).

Aktuální verze je 3.0, která vyšla na konci roku 2007 společně s Frameworkem 3.5 a Visual Studiem 2008. Předchozí verze 2.0 je z roku 2005 [5].



Obrázek 20: Architektura .NET Framework [16]

6.2.4 XML (Extensible Markup Language)

Jedná se o rozšířitelný značkovací jazyk vyvinutý a standardizovaný konsorciem W3C. Struktura XML dokumentu se skládá z deklarace, tagů, elementů a atributů. Používá se pro serializaci dat. Zpracování je podporováno v mnoha programovacích jazycích a nástrojích [4].

Správně strukturovaný dokument musí splňovat následující pravidla:

- Obsahuje pouze jeden kořenový element („root“).
- Každá hodnota atributu v elemetu musí být ohraničena uvozovkami.
- Každý neprázdný element („name“) musí být ohraničen startovací a ukončovací značkou.
- Jednotlivé elementy mohou být vnořeny do jiných elementů ale nesmí se překrývat.

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
  <element atribut="1">
    <name>Ukázka 1</name>
  </element>
  <element atribut="2">
    <name>Ukázka 2</name>
  </element>
</root>
```

Obrázek 21: Ukázka XML dokumentu

6.2.5 XML Schema Definition (XSD)

Tento dokument popisuje strukturu vytvářeného XML dokumentu. Pomocí něj lze kontrolovat:

- obsah dokumentu,
- správnost dat
- definovat počty elementů
- pořadí elementů
- zda musí element vždy obsahovat hodnotu
- datové typy elementů a atributů

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="element">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string" />
            </xs:sequence>
            <xs:attribute name="atribut" type="xs:int" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Obrázek 22: Ukázka XML schématu

6.2.6 DLL (Dynamic Link Library)

Jedná se koncept vytvořený společností Microsoft pro sdílené knihovny. Typicky knihovna obsahuje jednu nebo více funkcí, které mohou být použity v aplikacích. K funkcím se přistupuje buď statickým, nebo dynamickým odkazem na knihovnu. DLL knihovnu může využívat více aplikací současně.

6.2.7 Správa verzí

Při vývoji softwaru se mohou použít nástroje pro správu verzí vyvíjené aplikace. Uchovávání historie veškerých provedených změn se nazývá verzování. V programování se nejčastěji verzují zdrojové kódy aplikace, obecně lze verzovat jakékoliv soubory.

Systém správy verzí eviduje kdo, kdy a jak změnil jaký řádek zdrojového kódu. To slouží k přehlednému sledování změn a aktuálního stavu. Každé změně je přiděleno jedinečné číslo nazývané číslo revize. Pokud by se v aplikaci vyskytla chyba lze se kdykoliv vrátit k jakékoliv předchozí verzi.

Dalším využitím nástroje správy verzí je při vývoji aplikace více programátory současně. Jelikož jsou všechny změny hlídány, systém správy verzí zamezí kolizi při změně stejného kódu více programátory. Při větších projektech je verzování nezbytná součást vývoje. Nejznámějšími systémy jsou CSV, Subversion a SVN. Při vývoji byl použit verzovací nástroj Tortoise SVN.

7 Testování

Jedná se poslední fázi při vývoji aplikace. Testování je důležité pro ověření, zda program dělá to co má a demonstruje, že program neobsahuje žádné chyby.

V průběhu vývoje byly uvolňovány funkční verze, které byly ihned testovány při vytváření nových definic křižovatek. Testy by měly vždy provádět osoby, které aplikaci neprogramovaly. Testuje se každá akce programu a o výsledcích testu je veden podrobný protokol (Obrázek 23).

Datum: 16. 10. 2009

Provedl: Jan Novák

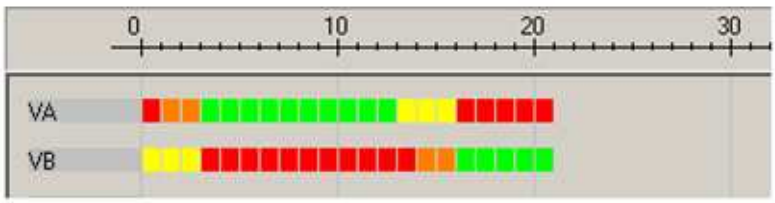
Kontroloval: Petr Novák

Popis funkce:
Délka Free uvádí dobu trvání základní zelené zvolené skupiny ve fázi signálního plánu, pokud není ovlivněna jinými pravidly, která ji mohou prodloužit (ne zkrátit).

1. Kontrola průběhu

Podmínky provedení:
Skupina VA 10 sekund v 1. fázi, VB 5 sekund ve 2. fázi

Výsledek - záznam monitorování přes CROSS_LTC



Výsledek:

Ve zdrojovém kódu ověřena funkčnost podle popisu ověřované funkce.

Závěr: Průběhy odpovídají nastaveným parametrům.

Obrázek 23: Protokol testu

8 Nasazení a distribuce

Posledním problémem, který se u nových aplikací musí vyřešit, je zajištění snadného nasazení aplikace u uživatele a jak aplikaci k uživateli doručit.

8.1 Číslování verzí aplikace

Verzí se označuje stádium vývoje aplikace. Nejčastěji je označováno číselnou hodnotou složenou ze čtyř částí. První číslo udává major verzi, (číslo hlavní verze), za tečkou následuje druhé číslo minor verze (vedlejší číslo verze), dále následuje číslo revize a poslední čtvrté číslo udává číslo sestavení. Verzi programu lze zapsat například takto: 0.9.2.30.

8.2 Nasazení

Pro chod aplikace je nezbytný .NET Framework 2.0 nainstalovaný na klientském počítači. Na hardwarovou konfiguraci aplikace není náročná. Aplikace byla testována jak pod operačním systémem Microsoft Windows XP, tak i pod novým operačním systémem Windows 7. Aplikace nevyžaduje stálé připojení k síti internet.

Po dokončení a testování aplikace začíná fáze předávání programu uživateli. Platforma .NET poskytuje vývojářům několik způsobů nasazení aplikace k uživateli, které budou popsány v následujícím textu

ClickOnce

Technologie ClickOnce umožňuje distribuci aplikace umístěním do sdílené složky na webovém serveru nebo na médiu jako je CD a paměťová karta ClickOnce řeší bezpečnostní problémy, kde uživatel musel mít práva správce pro instalaci běžné aplikace. Nyní uživateli postačují stejná oprávnění jako pro spuštění aplikace. ClickOnce řeší i automatickou aktualizaci bez potřeby většího zásahu uživatele.

Aplikace instalované pomocí technologie ClickOnce se neukládají do složky Program Files, ale do aplikačního úložiště, které se nachází ve složce Local Settings pod složkou uživatele v Documents and Settings. ClickOnce udržuje v úložišti aplikace všechny verze, které uživatel kdy použil. Taky se může vrátit k předchozí verzi. Po nainstalování je aplikace přidána do seznamu nainstalovaných aplikací v Ovládacím panelu. Aplikace instalovaná přímo ze sítě Internet, má automaticky nastavenou menší důvěryhodnost než klasické aplikace. Nemůže tedy přistupovat k systémovým souborům, pokud ale aplikace vyžaduje vyšší oprávnění je uživatel vyzván k povolení oprávnění.

Princip publikování aplikace pro nasazení pomocí technologie ClickOnce spočívá ve vytvoření dvou souborů ve formátu XML. První z nich je manifest pro aplikaci a druhý manifest nasazení. Visual Studio tyto soubory umožňuje jednoduše vytvořit pomocí funkce Publish, která všechny potřebné soubory uloží do složky určené k nakopírování na webový server [8].

V následující tabulce je základní porovnání dvou nejpoužívanějších způsobů nasazení [9].

	ClickOnce	**Windows Installer**
Automatický update	Ano	Ano
Po instalační – rollback	Ano	Ne
Aktualizace z webu	Ano	Ne
Neovlivňuje sdílené komponenty a jiné aplikace	Ano	Ne
Zaručená bezpečnost oprávnění	Oprávnění pouze pro aplikaci	Všechna práva zaručena
Požadavky na oprávnění	Internet nebo intranet	Administrátor
Podepsání aplikačního a nasazovacího manifestu	Ano	Ne
Uživatelské rozhranní instalace	Jednoduchý promt	Průvodce
Instalace Assemblies na požádání	Ano	Ne
Instalace sdílených souborů	Ne	Ano
Instalace ovladačů	Ne	Ano
Instalace do GAC	Ne	Ano
Instalace pro více uživatelů	Ne	Ano
Přidání zástupce do nabídky Start	Ano	Ano
Přidání do nabídky Po spuštění	Ne	Ano
Přidání zástupce do Oblíbených	Ne	Ano
Registrace typů souborů	Ne	Ano
Přístup do registrů v průběhu instalace	Omezený	Ano
Umístění aplikace	Aplikační úložiště ClickOnce	Složka Program Files

Tabulka 2: Porovnání ClickOnce a Windows Installer

Windows installer

Windows installer je služba starající se o instalace, aktualizace, opravy a odstraňování aplikací v operačních systémech Microsoft Windows. Instalátor má přístup do registrů, souborů na pevném disku, k zástupcům vytvořených na ploše a v nabídce start. Při odinstalování aplikace instalátor kontroluje, zda není na soubory odkazováno jinými aplikacemi. Pokud ano, instalátor soubory ponechá pro zajištění bezchybného provozu systému.

Vytvoření tohoto instalátoru rovněž podporuje vývojové prostředí Visual Studio. Je zapotřebí založit nový projekt nasazení Setup project, kde se definují akce, které se mají při instalaci provést. Je zde možnost vytvoření průvodce instalací. Celý instalátor je uložen v souboru ve formátu MSI. Pro instalaci je zapotřebí oprávnění správce. Aplikace se instaluje do složky Program Files [1].

Porovnání Windows installer a ClickOnce se nachází v tabulce 2.

8.3 Distribuce

Aplikace bude k zákazníkovi distribuována pomocí instalačního souboru vytvořeného pomocí windows installer 0. Uživatel si tento instalační soubor může stáhnout v zabezpečené sekci pro zákazníky webového serveru firmy.

Xcopy

Nasazování typu Xcopy označuje proces, kdy se zkopíruje sada souborů plně funkční aplikace do složky na cílovém počítači. Neprovádí se žádné zásahy do registu, ani do nastavení počítače. Toto sestavení není vhodné pokud chcete automatizovaně umístit ikonu zástupce do menu start, popřípadě na plochu nebo do položek oblíbených. Pokud je potřeba při nasazení využívat více funkčností, musí být použit jeden z instalátorů aplikace, které budou následně popsány [1].

9 Závěr

Cílem této práce bylo navrhnout a implementovat aplikaci pro definování řadičů světelných signalizačních zařízení. Aplikace umožní uživateli snadnou organizaci těchto definic podle měst a křižovatek.

Základní verze systému byla vyvinuta a je používána v praxi. Nadále se systém neustále zdokonaluje a funkčně rozšiřuje. Každá vytvořená nová verze aplikace je testována a posléze nasazena do praxe. Všechny požadavky kladené na funkčnost a bezpečnost aplikace byly splněny. Aplikace byla navržena a vyvíjena tak, aby její další rozšiřování bylo co nejjednodušší.

Díky této práci jsem si rozšířil znalosti nejen v oblasti Informatiky ale i v oboru dopravního inženýrství zaměřeného na světelná signalizační zařízení.

10 Literatura

[1] Christian Nagel, Bill Evjen, Jay Glynn, Morgan Skinner, Karli Watson, Allen Jones: C# 2005 Programujeme profesionálně. Computer Press 2006. ISBN 80-251-1181-4

[2] Microsoft: Double Buffered Graphics, URL: http://msdn.microsoft.com/en-us/library/b367a457.aspx, ze dne 4.4.2010

[3] Scott W. Ambler: UML 2 Use Case Diagrams, URL: http://www.agilemodeling.com/artifacts/useCaseDiagram.htm, ze dne 20.3.2010

[4] W3C: Extensible Markup Language (XML), URL: http://www.w3.org/XML/

[5] Jiří Gabriel: Bakalářská práce 2008

[6] W3C: XML Signature Syntax and Processing (Second Edition), URL: http://www.w3.org/TR/xmldsig-core/ ze dne 4.4.2010

[7] Microsoft: Overview of Globalization and Localization, URL: http://msdn.microsoft.com/en-us/library/Aa292205, ze dne 26.3.2010

[8] Borek Bernard: ClickOnce, URL: http://www.borber.com/files/IZI449-ClickOnce.pdf, leden 2006, ze dne 27.3.2010

[9] Microsoft: ClickOnce Deployment Overview, URL: http://msdn.microsoft.com/en-us/library/142dbbz4%28VS.80%29.aspx, ze dne 27.3.2010

[10] Microsoft: XML and the DataSet, URL: http://msdn.microsoft.com/en-us/library/84sxtbxh(VS.71).aspx, ze dne 28.3.2010

[11] Zákon č. 361/2000 Sb. (O silničním provozu) § 70 až § 74 (14.9. 2000) http://www.mdcr.cz/NR/rdonlyres/C33E38FC-A12A-46B8-9E89-841136D385FE/0/MicrosoftWord361.pdf, ze dne 24.3.2010

[12] Vyhláška č. 30/2001 Sb. § 24 (10.1.2001) http://www.mdcr.cz/NR/rdonlyres/DFE87D07-9E39-467D-95F9-35D3AB525369/0/MicrosoftWord30.pdf, ze dne 24.3.2010

[13] CROSS ZLÍN s.r.o.: Světelné signalizační zařízení s využitím technologií CROSS ZLÍN, 4/2009

[14] Červená, žlutá, zelená, (20.1.2009) URL:http://www.praha.eu/jnp/cz/home/doprava_v_praze/automobilova/cervena_zluta_zelena.html, ze dne 2.4.2010

[15] Tomáš Bosák: ADO.NET (4. 9. 2006), URL: http://programujte.com/?akce=clanek&cl=2006090202-ado-net, ze dne 20.4.2010

[16] Microsoft: Moving Java Applications to .NET, URL: http://msdn.microsoft.com/en-us/library/ms973842.aspx, ze dne 20.4.2010

A. CD-ROM

- Elektronická verze diplomové práce